

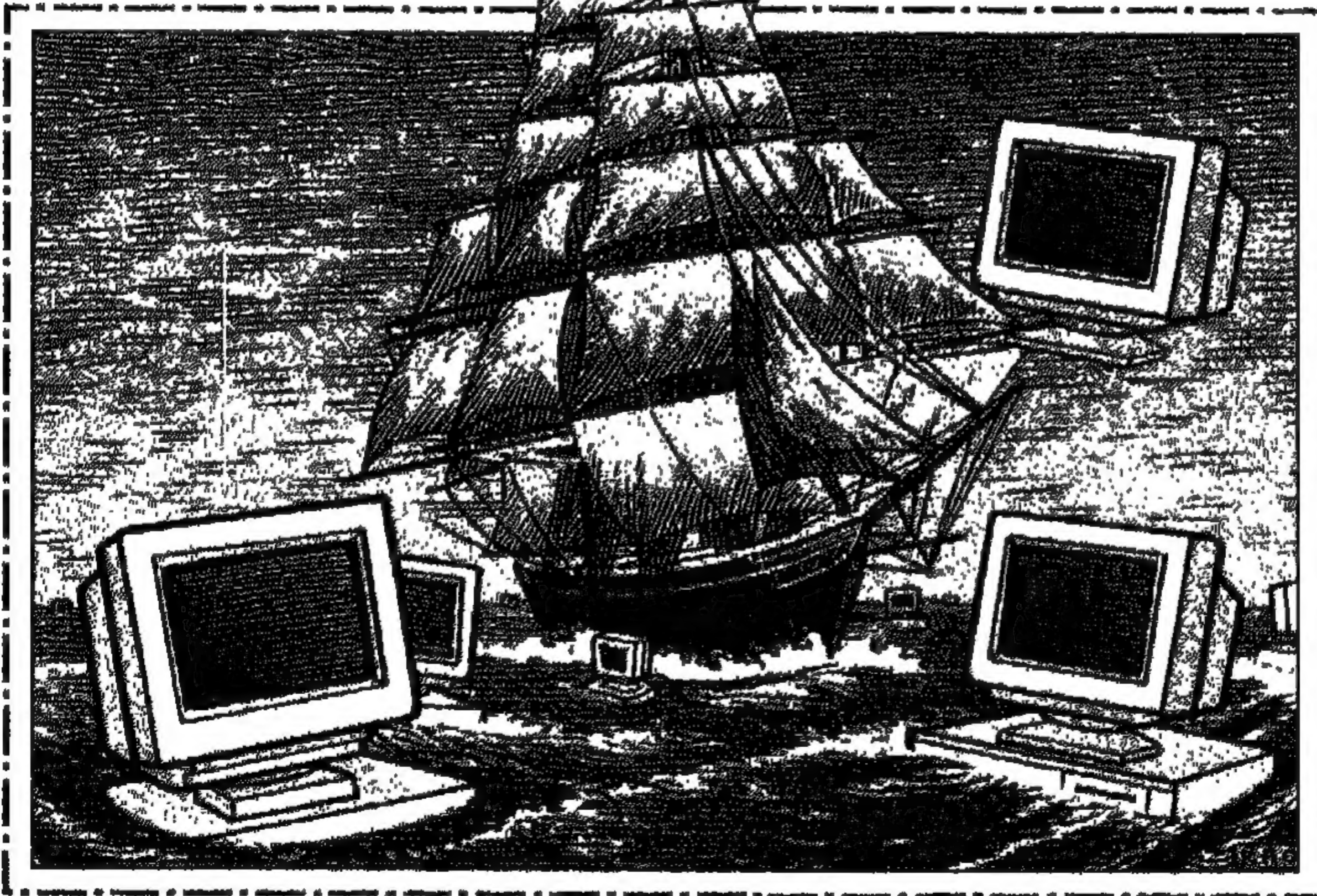


محاضرات كليبر

Clipper Course Notes

الجزء الثالث: البرمجة المتقدمة

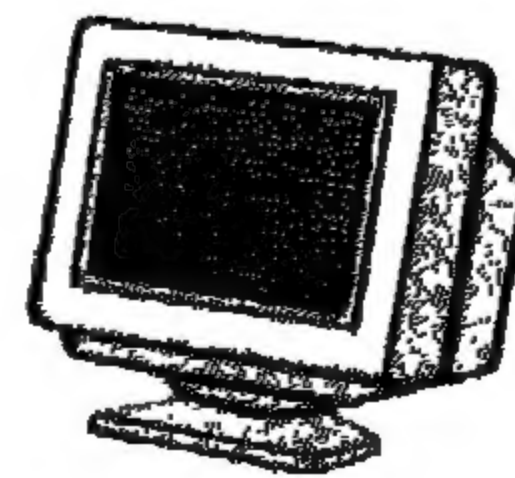
جديد!
الإصدار 5.2



كتاب



قرص



الجمعية العلمية

للدراسات والبحوث

في علوم الحاسوب

والهندسة

والرياضيات

والفيزياء

والكيمياء

والبيولوجيا

والطب

والفنون

واللغات

والعلوم الإنسانية

والعلوم الاجتماعية

والعلوم التطبيقية

والعلوم العسكرية

والعلوم السياسية

والعلوم الاقتصادية

والعلوم الإدارية

والعلوم القانونية

والعلوم الشرعية

والعلوم الدينية

والعلوم التاريخية

والعلوم الجغرافية

والعلوم البيئية

والعلوم الصحية

والعلوم الزراعية

والعلوم الصناعية

والعلوم العسكرية

والعلوم السياسية

اهداءات 2002

المهندس / سيد مصطفى أبو السعود

القاهرة

Nc
005.3

M469

V3 محاضرات كليبر

Clipper Course Notes

الجزء الثالث: البرجة المتقدمة

005.3

سليمان بن عبد الله الميمان - المحاضر

الأستاذ/أحمد فراس مهاني

الدكتور/محمد سعيد دباس

النشر والتوزيع:

٢١


الميمان للنشر والتوزيع


BIBLIOTHECA ALEXANDRINA
مكتبة الاسكندرية

ص.ب: ٩٠٠٢٠ - الرياض ١١٦١٣

هاتف: ٤٠٢١٢١٩ - ٤٠٢٦١٩٤

فاكس: ٤٠١٤٩٩٦


BIBLIOTHECA ALEXANDRINA
مكتبة الاسكندرية
كتب عربي
(أمدات)

رقم التسجيل ٧٥١٩٨

محاضرات كليبر 5.2: البرمجة المتقدمة

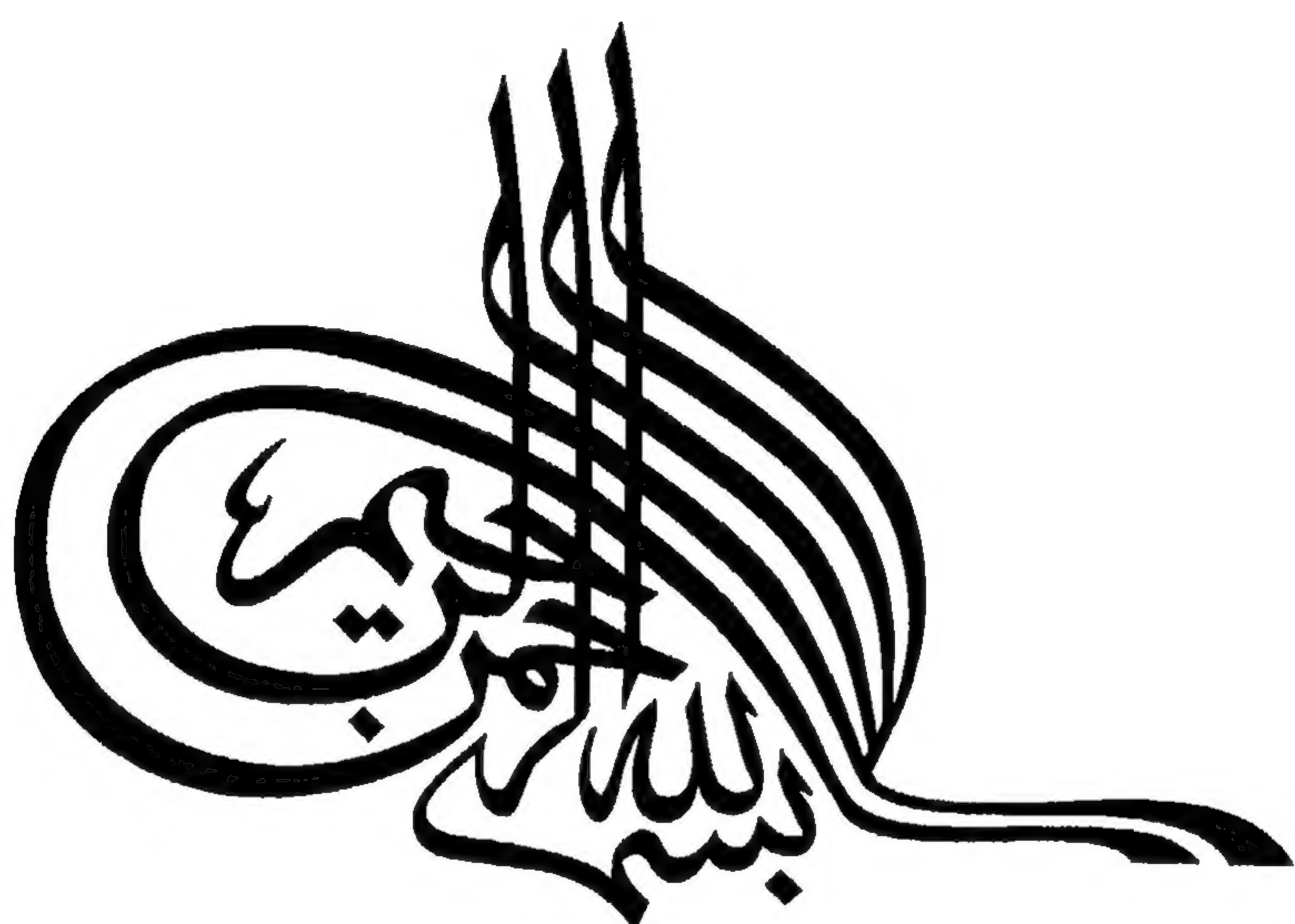
الطبعة الأولى - الرياض - ١٤١٥ هـ

حقوق الطبع محفوظة

حقوق الطبع والنشر محفوظة لدار الميمان للنشر والتوزيع، ولا يحق لأي شخص نشر هذا الكتاب أو أي جزء منه أو تصويره أو إعادة طبعه أو تخزين محتوياته أو نقلها بأي وسيلة إلا بعد الحصول على إذن خطي وصريح مكتوب من الناشر.

تنويه

تم إعداد هذه المحاضرات بالتعاون مع مؤسسة جرمبفيس الأمريكية المتخصصة في إعداد برامج تعليم لغة كليبر. وهذه المؤسسة معتمدة من قبل شركة Computer Associates، المالك الرسمي لجمع لغة كليبر 5.X.



المحتويات

المحتويات

القسم الأول: (الاستعراض TBCOLUMN و TBROWSE).....	١٥
هدف جدول الاستعراض الأول TBrowse Object.....	١٩
الطريقة السهلة لضبط حجم النافذة.....	٢٤
الوظيفة b:stable و b:stabilize().....	٢٥
كيانات منفصلان.....	٢٧
معالجة الضغط على المفاتيح العامة.....	٢٨
تشغيل المفاتيح الإضافية.....	٣١
إلغاء الإجراءات الافتراضية.....	٣١
زيادة حمل الإجراءات الافتراضية.....	٣٢
إنشاء أهداف "عمود استعراض الجداول" TBColumn.....	٣٣
الكتلة BLOCK.....	٣٤
الشحنة Cargo.....	٣٥
كتلة الألوان ColorBlock.....	٣٥
إيقاف عمل كتلة الألوان colorBlock.....	٣٨
فاصل الأعمدة COLSEP.....	٤١
تحديد الألوان defcolor.....	٤٢
التذييلة Footing.....	٤٣
فاصل التذييلة FootSep.....	٤٤
الترويسة Heading.....	٤٥
فواصل الترويسة headSep.....	٤٦
الصورة Picture (أضيف في الإصدار ٥,٢).....	٤٧
العرض Width.....	٤٩
في أعماق أهداف استعراض الجداول TBrowse Objects.....	٥٢
المتغيرات الفورية Instance Variables.....	٥٢
التظليل الآلي autoLite (لا يمكن تعيينه).....	٥٣

٥٤	الشحنة cargo (يمكن تعيينه).....
٥٤	عدد الأعمدة colCount.....
٥٤	جدول الألوان ColorSpec (يمكن تعيينه).....
٥٦	موضع المؤشر في العمود colPos (يمكن تعيينه).....
٥٧	فاصل الأعمدة colSep (يمكن تعيينه).....
٥٨	فاصل التذييل footSep (يمكن تعيينه).....
٥٩	الإقفال freeze (يمكن تعيينه).....
٦١	كتلة الانتقال إلى الأعلى وإلى الأسفل (يمكن تعيينها).....
٦٢	فواصل الترويسة headSep (يمكن تعيينه).....
٦٤	تجاوز الحد الأسفل والحد الأعلى.....
٦٥	عرض العمود الأيسر leftVisible وعرض العمود الأيمن rightVisible.....
٦٨	إحداثيات الاستعراض nBottom , nleft , nRight , nTop.....
٧٠	عداد الصفوف rowConnt.....
٧٠	موضع المؤشر في الصف rowpos.....
٧٣	كتلة التجاوز Skipblock.....
٧٨	ثبات stable.....
٧٨	وظائف TBrowse ذات الأغراض الخاصة.....
٧٩	وظيفة إضافة عمود addColumn().....
٧٩	وظيفة تلوين المستطيل colorRect().....
٨٣	وظيفة "عرض العمود" colWidth().....
٨٤	وظيفة "التهيئة" configure().....
٨٤	وظيفة Hilite() و deHilite().....
٨٥	وظيفة "حذف عمود" delColumn().....
٨٥	التثبيت الجبري forceStable().....
٨٦	استرجاع هدف عمود getColumn().....
٨٦	إدراج عمود insColumn().....
٨٦	الوظيفة Invalidate().....
٨٧	وظيفة refreshAll() و refreshCurrent().....
٩١	وظيفة "ضبط الأعمدة" setColumn.....
٩٤	نسخ العمود CloneColumn().....
٩٦	وظيفة "التثبيت" stabilize().....
٩٧	أمثلة متقدمة على ميزات الاستعراض TBROWSE.....

استخدام "الشحنة" Cargo لمسح حقول حرفية عريضة	١١٠
استعراض مصادر بيانات بديلة	١١٥
استعراض مصفوفات بسيطة	١١٥
استعراض مصفوفات متداخلة	١١٦
استعراض مصفوفات متداخلة بطول غير معروف	١١٨
استعراض حقول المذكرة Memo Fields	١٢١
استعراض ملفات Btrieve	١٢٣
جداول استعراضات متعددة متزامنة	١٢٧
الاستعراضات الرئيسة/الفرعية	١٣١
جدول الاستعراضات الرئيسة/الفرعية/التابعة للفرعية	١٤٠
الانتقال من مجموعة جزئية إلى أخرى	١٥٠
أهداف الاستعراض الساكنة Static TBrowse Objects	١٥٦
تحسين الانزلاق العمودي في جدول TBrowse	١٦٢
الاستعراض Browse متعدد الصفوف (القسم الأول)	١٦٩
نقاط هامة	١٧١
القسم الثاني (أهداف GET/نظام GET)	١٧٧
تمهيد	١٧٩
تحليل أمر @..GET	١٨١
عبارات @..get في كليبز الجديد	١٨٣
ميزة التلوين التلقائي	١٨٤
مصفوفة GETLIST	١٨٥
عمليات القراءة المتداخلة NESTED READS	١٨٧
استهلال أمر GET (كليبز ٥,٢ فقط)	١٨٩
تحسين نظام إدخال البيانات	١٩١

١٩٢	الطريقة الثانية: أفضل (GETLIST)
١٩٣	الطريقة الثالثة: الأفضل (STATIC GETLIST)
١٩٥	عبارة WHEN
١٩٥	تجاوز أمر GET
١٩٦	رسائل المساعدة باستخدام عبارة WHEN
١٩٦	إدخال البيانات باستخدام WHEN
١٩٨	استخدام المفاتيح السريعة مع WHEN
١٩٩	استخدام GETLIST مع عبارة WHEN
٢٠١	تعميم الشيفرة الخاصة بك
٢٠١	المثال الأول: الحصول على متغيرات محلية
٢٠٢	المثال الثاني: القراءة المتداخلة مع وظيفة GETACTIVE()
٢٠٥	حفظ أوامر GET باستخدام "المخزن" STACK
٢٠٧	شاشات إدخال بيانات متعددة الصفحات
٢١١	الوظيفة GETNEW()
٢١٣	المتغيرات الفورية لهدف GET
٢١٤	المتغير الفوري "تاريخ غير صحيح" badDate
٢١٤	الكتلة block (يمكن تعيينها)
٢١٥	ذاكرة مؤقتة buffer (يمكن تعيينه)
٢١٥	الشحنة cargo (يمكن تعيينه)
٢١٥	التغيير changed (يمكن تعيينه)
٢١٦	العمود col (يمكن تعيينه)
٢١٧	طيف الألوان colorSpec (يمكن تعيينه)
٢١٨	موضع الفاصلة العشرية decPos
٢١٨	حالة الخروج exitState
٢١٩	استخدام exitState للانتقال من هدف GET إلى آخر
٢٢٣	مظلل hasFocus
٢٢٤	الاسم name (يمكن تعيينه)
٢٢٥	القيمة الأصلية original
٢٢٥	الصورة picture (يمكن تعيينه)

٢٢٦	موضع المؤشر POS (يمكن تعيينه)
٢٢٧	كتلة لاحقة postBlock (يمكن تعيينه)
٢٢٨	كتلة سابقة preBlock (يمكن تعيينه)
٢٣٠	القارئ reader (يمكن تعيينه)
٢٣٠	المتغير الفوري "مرفوض" rejected
٢٣٠	المتغير الفوري "الصف" row (يمكن تعيينه)
٢٣١	المتغير الفوري "الرمز الفرعي" subscript
٢٣٣	المتغير الفوري "النوع" type
٢٣٤	متغير "الخروج من الذاكرة" typeout
٢٣٥	تشغيل المتغيرات الفورية
٢٣٥	المسألة
٢٣٦	الحل
٢٣٦	استخدم مصفوفات بدلاً من متغيرات الذاكرة
٢٣٧	أهداف GET غير المرئية
٢٣٧	تعديل الصورة Picture
٢٣٧	إنقاص عبارة WHEN
٢٤١	أهداف GET مشغلة بواسطة البيانات DATA-DRIVEN
٢٤٢	الصيغة Formulae
٢٤٣	قيم محلية مستقلة Detached Locals
٢٤٣	الكبسلة Encapsulation
٢٥١	والوظائف المرتبطة بـ: GET
٢٥٢	عرض الألوان ColorDisp()
٢٥٥	تحسين أداء أمر @.GET
٢٥٥	البنية الهيكلية لملف GETSYS.PRG
٢٥٥	القارئ GetReader(<oGet>)
٢٥٦	استخدام وظيفة المفاتيح GetApplyKey(<oGet> , <nKey>)
٢٥٦	التدقيق السابق GetPreValidate(<oGet>)

٢٥٧	التدقيق اللاحق (<oGet>) GetPostValldate
٢٥٧	ضبط المفاتيح (<oGet>) GetDoSetKey
٢٥٨	قراءة ملف الشاشة (<bFormat>) READFORMAT
٢٥٨	إنهاء القراءة (<lKill>) READKILL
٢٥٩	تحديث القراءة (<lUpdated>) READUPDATED
٢٥٩	كتابة وظيفة (<lReader>) GetReader بديلة
٢٦٣	التوسع في استخدامات وظائف المفاتيح GetApplyKey0 البديلة
٢٦٨	استخدام وظيفة (<lMyReader>) MyReader المعدلة
٢٦٩	كتابة وظائف المفاتيح العادية
٢٧٠	إدخال كلمة السر (<lKeyPass>) GKeyPass
٢٧١	مفاتيح الحروف الملائمة (<lKeyProper>) GKeyProper
٢٧١	الوظيفة (<lKeyStep>) GKeyStep
٢٨١	قراءة موقوتة
٢٨٢	وظيفة توقيت الخروج (<lGFTimeOut>) GFTimeOut
٢٨٣	موجه المعالج الأولي Preprocessor Directive
٢٨٤	التدقيق الكلي
٢٨٤	مثال
٢٨٨	وظيفة GET العامة
٢٨٩	المتغيرات المطلوبة
٢٨٩	المتغيرات الاختيارية
٢٩٠	إعادة اقيمة
٢٩٠	ملاحظات
٢٩١	عبارة VALID
٢٩٤	ملاحظة
٢٩٥	الخلاصة
٢٩٩	معالجة الأخطاء والأهداف الخاصة بالأخطاء
٢٩٩	تمهيد
٢٩٩	برنامج ERRORSYS.PRГ في كليبر Summer'87
٣٠٠	بداية التسلسل..نهاية التسلسل
٣٠١	عبارة "إصلاح" RECOVER

٣٠٢	استخدام عبارة RECOVER
٣٠٣	برنامج ERRORSYS.PRG في كليب ٥,٠
٣٠٣	هدف الخطأ Error Object
٣٠٧	كتابة برنامجنا الخاص لمعالجة الأخطاء
٣١٠	ربط معالجات الأخطاء ببعضها
٣١١	أمثلة على معالج الخطأ العادي
٣١١	مفتاح فهرس غير صحيح Invalid Index Key
٣١٢	ملفات مفقودة Missing Files
٣١٣	خطأ "تجاوز الحد" في جدول الاستعراض TBrowse
٣١٤	التحقق من حالة إقفال السجل الحالي
٣١٤	تقييم كتل الشيفرة المعرفة من قبل المستخدم
٣١٥	إنشاء أهداف الخطأ باستخدام الوظيفة ERRORNEW()
٣١٦	وظيفة التنبيه ALERT()
٣١٨	وظيفة معالج الخطأ ErrorHandler() (جديد في إصدار كليب ٥,٢)
٣١٩	تسجيل الأخطاء في الأسطوانة
٣٢٠	الخلاصة

القسم الأول

جدول الاستعراض TBrowse

عمود الاستعراض TBColumn

مُهَيِّدٌ

يقدم كليب 5.x عدداً كبيراً من الميزات الجديدة القوية ، مثل المعالج الأولي preprocessor ، والإمكانات الهائلة للمصفوفات arrays ، وكتل الشيفرة code blocks . إلا أن هذه الإضافات العظيمة تبدو ضئيلة بالمقارنة مع أعظم ميزة جديدة وهي : فئات الهدف object classes .

يقدم كليب 5.x أربع فئات هدف object classes، يمثل مجرد وجودها طريقة مختلفة تماماً عما اعتدت عليه في قواعد البيانات (أو الإصدارات السابقة في كليب). وكلما ازدادت معرفتك بفئات الهدف في كليب 5.2 ، تغير أسلوبك في البرمجة بشكل ملحوظ وخاصة في اثنتين من هذه الفئات ، اللتين لهما تأثير كبير وهما: فئة استعراض الجداول TBrowse وهي مفيدة جداً في عملية استعراض البيانات ، وكذلك عمود استعراض الجداول TBColumn الذي يحدد نوع البيانات التي سيستعرضها هدف "استعراض الجداول" TBrowse .

يركز هذا البحث على استعراض "الجداول" TBrowse و "عمود استعراض الجداول" TBColumn من بين فئات الهدف ، وعلى كيفية تفاعلها مع بعضهما . وبانتهاء البحث ستتعرف على كافة المتغيرات الفورية المحدثة بشكل كامل ، وعلى الطرق المتعلقة بهاتين الطبقتين. وستشاهد أمثلة عملية عن "استعراض الجداول" لاستعراض المصفوفات البسيطة والمتداخلة ، وقواعد بيانات ، وحتى حقول الذاكرة. ويمكنك إدخال العديد من هذه الأمثلة كما هي في برامجك التطبيقية ، إلا أنه من الأفضل استخدام الأمثلة كمنطلق لتجهيز تشكيلات خاصة بك من TBrowse.

هدف جدول الاستعراض الأول TBrowse Object

من المعروف أن الناس لا يحبون التغيير. فنحن نحب المؤلف (مثل الوظيفة (DBEDIT)) ونخشى المجهول (مثل فئة الهدف TBrowse) علماً بأن لـ TBrowse إمكانيات خيالية رائعة غير محدودة ، لكنه يتطلب بعض التعليم والتجربة لتسخير هذه الطاقات. ومع ذلك فلا تقلق ، فإن تجهيز TBrowse واستخدامه في الاستعراض يتطلب جهداً قليل فقط.

لتوضيح ذلك ، لننشئ بسرعة هدف TBrowse يستعرض محتويات أية قاعدة بيانات.

```
function tbrow01(cDbfname)
local nField
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
scroll()
use (cDbfname) new
for nField := 1 to fcount()
    oColumn := TBColumnNew(field(nField), fieldblock(field(nField)))
    oBrowse:AddColumn(oColumn)
next
do while ! oBrowse:stabilize()
    inkey(.5) // this is only to show row by row display... you
              // won't generally want anything inside this loop
enddo
inkey(0)
use
return nil
```

أرايت! لقد تطلب هذا العمل حوالي خمسة عشر سطراً من البرمجة (وليست جميعها ضرورية).

والآن دعنا نراجع العملية بشيء من التفصيل TBrowse:

١- لقد جهزنا هدف TBrowse باستخدام الوظيفة TBrowseDB(). وفي هذه المرحلة فإن هدف TBrowse هو بمثابة صيغة فارغة ليس إلا. اتبع القاعدة اللغوية الخاصة باستدعاء هذه الوظيفة هي:

TBrowseDB() (<nTop> , <nLeft> , <nBottom> , <nRight>)

جميع هذه المتغيرات رقمية وتتوافق مع إحداثيات الشاشة لاستخدامها في الاستعراض. لاحظ وجود وظيفة أخرى هي TBrowseNew() يمكن استخدامها لإنشاء أهداف TBrowse (وسنبحث الفرق بين TBrowseDB() و TBrowseNew () في الوقت المناسب).

٢- إن الهدف TBrowse حتى هذه المرحلة هو هيكل فارغ ، ليس إلا . لذا فإن علينا أن نملأه بالأعمدة ليتمكن الاستفادة منه. ونقوم بإنشاء هدف "عمود استعراض الجداول" TBColumn لكل حقل في قاعدة البيانات باستخدام الوظيفة TBColumnNew(). أما القاعدة اللغوية الخاصة بوظيفة TBColumnNew() هي:

TBColumnNew(<cHeading> , <bBlock>)

حيث أن <cHeading> هي سلسلة حرفية ستستخدم كتروية للعمود. فإن لم تكن ترغب باستخدام تروية للعمود ، فإما أن تمرر سلسلة فارغة (' ') أو تجاوز المتغير الأول بأكمله. أما الوظيفة FIELD(x) فهي تعيد السلسلة الحرفية التي تمثل اسم الحقل في موقع معين (وليكن x) في بنية قاعدة البيانات ، بحيث تلبي الحاجة لمتغير التروية <cHeading>.

أما <bBlock> فهي كتلة شيفرة تحدد عند تقييمها وتحديد محتويات هذا العمود. وفي المثال أعلاه تستخدم الوظيفة FIELDBLOCK() لإنشاء كتلة الشيفرة لاستعادة كل حقل في قاعدة البيانات.

٣- تتم إضافة كل عمود إلى الهدف TBrowse عن طريق الوظيفة `addColumn()`. لاحظ أن القاعدة اللغوية لتنفيذ هذه الطريقة مشابهة تماماً لاستدعاء الوظائف بل إن بل هي بحد ذاتها وظيفة مخصصة لفئات هدف محددة. والفرق الوحيد بينها هو أنك عندما تنفذ وظيفة ما من هذا النوع عليك أن تستهلها بإشارة إلى الهدف يليها عامل الإرسال (" : "). وإذا لم تورد هذه الإشارة ، فسيستنتج كليبز بأنك تستدعي وظيفة من الوظائف المعتادة ، وهذا يعني عدم حصولك على النتائج المرجوة.

سنشير في هذا البحث إلى طرق (هو الاسم الخاص بوظائف فئات الهدف) TBrowse بوضع "b:" قبلها.

٤- يتم عرض البيانات في هدف TBrowse باستدعاء الوظيفة `b:stabilize()`. وقد سميت بذلك بدهياً ، ولكن هذه هي الآلية الرئيسية لعرض البيانات في نافذة TBrowse.

تختلف هذه الوظيفة اختلافاً جذرياً عن أسلوب `DBEDIT()` الذي تستدعي به وظيفة تعرض شاشة ممتلئة بالبيانات وتنتظر أن تقوم بضغط أحد المفاتيح. إن TBrowse يختلف تماماً ، فعندما تقوم بإنشاء هدف TBrowse يكون لكل صف في نافذة البيانات مؤشر منطقي مطابق يقوم بمتابعة البيانات والتأكد من عرضها بشكل ملائم. ويتم عند الإنشاء ضبط جميع هذه المؤشرات في حالة غير حقيقي (F.) لأن أياً من البيانات لم يتم عرضها.

تعرض الوظيفة `b:stabilize()` صفاً واحداً من البيانات على حدة في كل مرة، وتضبط المؤشر الخاص بهذا الصف في حالة حقيقي (T.) للدلالة على أنه تم عرضه بشكل صحيح. وتقوم الوظيفة `b:stabilize()` بإعادة القيمة المنطقية : حقيقية (T.) وإذا تم عرض كافة البيانات في نافذة TBrowse بشكل صحيح ، أو غير حقيقي (F.) إذا لم تعرض بشكل صحيح. ينبغي ، لهذا السبب ، تكرار استدعاء

الوظيفة (b:stabilize) حتى تعود القيمة حقيقية (T.). وفي حال إستدعائها مرة واحدة فقط ، فسيعرض على الشاشة صف واحد فقط من البيانات.

لاحظ أنه عندما تستدعي الوظيفة (b:stabilize) ، فسيعرض أيضاً ترويسات وتذييلات للأعمدة ، وفواصل ترويسة / تذييل (إذا كنت تستخدمها).

هذا كل مايتعلق بكتابة هدف TBrowse. وهل تلاحظ أنه أسهل مما كنت تظن؟ إلا أن هدف TBrowse هذا لايقوم بأعمال كثيرة. فإنه يقوم فقط بعمل الشاشة بسجلات من قاعدة البيانات وبتنظر الضغط على أحد المفاتيح ثم ينهي العملية.

ينبغي قبل كل شيء إضافة معالجة آلية أولية لضغط المفاتيح. ولنضيف مساندة لمفاتيح الأسهم (للاتقال ضمن قاعدة البيانات) ومفتاح الخروج [Esc] (للخروج من TBrowse).

```
#include "inkey.ch"

function tbrow02(cDbfname)
local nCurrfield
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nFields
local cField
local nKey
scroll()
use (cDbfname) new
nFields := fcount()
for nCurrfield := 1 to nFields
    cField := field(nCurrfield)
    oColumn := TBColumnNew(cField, fieldblock(cField))
    oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
    do while ! oBrowse:stabilize()
    enddo
    nKey := inkey(0)
    do case
        case nKey == K_UP
            oBrowse:up()
        case nKey == K_DOWN
```

```
oBrowse:down()
case nKey == K_PGUP
  oBrowse:pageUp()
case nKey == K_PGDN
  oBrowse:pageDown()
case nKey == K_LEFT
  oBrowse:left()
case nKey == K_RIGHT
  oBrowse:right()
case nKey == K_ALT_F1 // toggle 25/50 line mode
  if maxrow() > 24
    setmode(25, 80)
  else
    setmode(50, 80)
  endif
  oBrowse:nBottom := maxrow()
endcase
enddo
use
return nil
```

هذا أفضل من ذي قبل ! فكما أن وظيفتي (b:stabilize) و (b:addColumn) طريقتان لـ TBrowse فإن الوظائف (b:up) و (b:down) و (b:right) و (b:left) كذلك. ويمكنك بالطبع تصور مايمكنها فعله ، وستجد لاحقاً أن هناك الكثير من طرق المعالجة الآلية الأخرى لضغط المفاتيح. فليس عليك كتابة أي من هذه الطرق ، بل إنها جميعها متضمنة في تعريف فئة الهدف TBrowse بانتظار استدعائك لها.

قد يبدو لك أن تكتب CASE الخاص بهذه لمعالجة الآلية الأولية لضغط المفاتيح لأول وهلة يتطلب جهداً كبيراً إذا ما قورن بوظيفة مثل DBEDIT() التي تعالج هذه الأعمال بشكل آلي. ولكن الجانب الإيجابي لهذه العملية هو أنك تكتسب "السيطرة التامة". ففي حال الرغبة في إيقاف عمل مفاتيح معينة أثناء الاستعراض فما عليك إلا أن تسقطها من العملية المنطقية للمعالجة الآلية لضغط المفاتيح ، ولن تحتاج إلى استخدام "ضغط المفاتيح" SET KEY لإيقاف عملها. (يمكنك أيضاً عكس وظائف المفاتيح حسب ما يناسب المستخدمين ، كأن تجعل سهم الاتجاه إلى أعلى يتجه إلى أسفل وسهم اليسار يتجه إلى اليمين). والأهم من ذلك أنك ستصل إلى درجة من الخبرة في عملية

المعالجة الآلية لضغط المفاتيح بحيث تكتبها مرة واحدة وتضمنها في وظيفة تستخدم في كافة تشكيلات TBrowse. وإن الراحة الأكيدة في النهاية تستحق حقاً هذا الجهد.

الطريقة السهلة لضبط حجم النافذة

بالضغط على مفتاحي **[F1] + [Alt]** ، وفي المثال ، يمكنك الاختيار ما بين ٢٥ و ٥٠ سطراً باستخدام وظيفة ضبط الحالة في كليبر 5.x (**SETMODE**) . ولقد أوردنا هذه الوظيفة كقاعدة منطقية لتوضيح الفارق الأساسي بين البرمجة المكيفة وفق الهدف (TBrowse) والبرمجة الإجرائية: وظيفة (**DBEDIT**) .

إذا سبق أن حاولت ضبط نافذة وظيفة (**DBEDIT**) فلا بد أنك تعرف صعوبة هذه العملية حيث تحتاج أولاً إلى أربعة متغيرات لمتابعة إحداثيات النافذة. ثم عليك أن تقوم ببرمجة معقدة للإستمرار بالخروج والدخول إلى وظيفة (**DBEDIT**) لأن هذه المتغيرات تغير قيمها.

أما باستخدام TBrowse فإن العملية أسهل بكثير ، لأن هدف TBrowse يعرف دائماً إحداثياته بفضل وجود المتغيرات الفورية **nTop** و **nLeft** و **nBottom** و **nRight** . وما علينا بعد استدعاء الوظيفة (**SETMODE**) ، إلا إعادة تعيين القيمة الجديدة الناتجة عن الوظيفة (**MAXROW**) إلى **b:nBottom** . وعندما نرجع إلى القاعدة المنطقية للتوازن فستعرف TBrowse بأن عليه أن يعرض الصفوف الإضافية.

الوظيفة () b:stabilize و b:stable

هذه ميزة أخرى تمتاز بها TBrowse عن الوظيفة DBEDIT(). وهي أنه يتم عرض البيانات باستدعاء الوظيفة () stabilize مرات وتكراراً. وإن إجراء أي تغيير بسيط في عبارة DO WHILE سيؤثر تأثيراً كبيراً على طريقة عرض البيانات.

```
do while ( key := inkey() ) == 0 .and. ! b:stabilize( )
enddo
```

أمامك الآن حالتنا خروج من هذه الحلقة: (أ) ضغط أحد المفاتيح ، أو (ب) التوازن العادي لعرض البيانات. وقد تتساءل عن معنى هذا. هل سبق أن رغبت أن تشاهد سجلاً ما في شاشة البيانات العاشرة؟ هل تذكر كم كانت هذه العملية مملة باستخدام الوظيفة DBEDIT() ؟ حتى لو ضغطت على مفتاح [PgDn] فستعرض لك وظيفة DBEDIT() ببطء بيانات الشاشة بأكملها قبل الانتقال إلى الشاشة التالية.

بينما باستخدام TBrowse والوظيفة () b:stabilize بمجرد أن تضغط على مفتاح [PgDn] تتوقف العملية مباشرة وتنتقل إلى شاشة البيانات التالية. وهذا وحده يفيد في توفير وقت طويل وثمين من وقتك ووقت المستخدمين.

للاستفادة من العملية المختصرة هذه بشكل فعال أكثر سنقوم بإضافة قاعدة

منطقية صغيرة بعد حلقة DO WHILE:

```
do while ( key := inkey() ) == 0 .and. ! b:stabilize( )
enddo
if b:stable                // or alternatively test for key == 0
endif
do case
    etc . . .
```

في حالة انقطاع عملية البيانات بضغط أحد المفاتيح لا داعي طبعاً للانتظار حتى يتم ضغط مفتاح آخر وبذلك يمكنك فحص حالة المتغير الفوري b:stable. إن المتغيرات

الفورية هي البيانات المنقولة من كل هدف ، وتمثل حالة هذا الهدف كما ورد سابقاً عند ذكر إحداثيات نافذة TBrowse.

ويمكنك التأكد من قيمتها في أي وقت ، كما يمكن تعيين قيم للعديد من المتغيرات الفورية. كما هو الحال مع الطرق الوارد ذكرها أعلاه ويجب عليك عند الإشارة للمتغيرات الفورية أن تستهلها باسم هدف ما ، يليه عامل الإرسال (":"). وإذا لم تورد هذه الإشارة ، فسيستنتج كليبر خطأً بأنك تبحث عن متغير قديم بسيط ، ولن تحصل على النتائج المرجوة.

سنشير في هذا البحث إلى المتغيرات الفورية TBrowse و TBColumn بوضع "b:" و "c:" قبلهما على التوالي.

لا بد أنك عرفت الآن أن التغير الفوري b:stable يرتبط ارتباطاً وثيقاً بطريقة b:stabilize() . مثل () b:stabilize حيث أن b:stable تعيد القيمة المنطقية استناداً إلى عرض كافة البيانات الممكنة بشكل صحيح ، وفي هذه الحالة يشتمل b:stable على قيمة "حقيقي" (T.) ، وإلا فسيحتوي قيمة "غير حقيقي" (F.) . إن الفرق الوحيد ، في الواقع بين b:stable والوظيفة () b:stabilize هو أن الأخيرة تقوم بعمل ما ، وهو محاولة إعادة عرض صف آخر من البيانات في نافذة TBrowse . وفي هذه الحالة ، فمن الأفضل تسجيل القيمة دون تنفيذ أي إجراء آخر.

لذلك ، إذا كانت نتيجة فحص b:stable "حقيقي" ، فإنك ستعلم بأن عملية عرض البيانات التي بدأتها الوظيفة () b:stabilize لم تنقطع ، وأن عليك أن تضغط مفتاحاً لتنفيذ إجراء ما.

فكرة مفيدة

يمكن استبدال نتيجة فحص b:stable بفحص المتغير بالضغط على أحد المفاتيح والاستمرار بالضغط للتأكد من أنها مازالت تساوي الصفر. وإذا كانت كذلك فإن هذا يشير إلى أن حلقة التوازن لم تنقطع بالضغط على المفتاح ، سيستخدم هذا النوع من الاختيار في الأمثلة اللاحقة لأنه أسرع من إرسال رسالة b:stable.

ملاحظة خاصة بمستخدمي كليب 5.2

أضيفت الوظيفة (b:forceStable) في الإصدار 5.2 من كليب ، وهي مساوية لحلقة التوازن التقليدية ، لكنها تنفذ العمليات بشكل أسرع بقليل. فإذا كنت تستخدم كليب 5.2 ولا تريد إيقاف عملية التوازن بالضغط على مفتاح ما ، فيمكنك استبدال حلقة التوازن باستدعاء فردي للطريقة (b:forceStable). ومع ذلك ، يكون لهذه العملية الأفضلية على إمكانية الانقطاع بالضغط على مفتاح ما كما ورد سابقاً.

كيانات منفصلان

من المهم قبل متابعة البحث ، التأكد من أن TBrowse ومصدر البيانات الخاص بك (ملف قاعدة البيانات DBF ، أو المصفوفة ، أو أيًا كان) كيانات منفصلان تماماً. بحيث أن TBrowse يقوم فقط بدور الاتصال البيئي مع البيانات. وهذا يعني أنه إذا كنت تقوم بتعديل بيانات على الشاشة فلن يعاد عرضها تلقائياً ، فليس من وظائف TBrowse (ولا يجب أن تكون في بنائها) المتابعة والتأكد من أن البيانات الموجودة في مصدر البيانات الأساسية قد تغيرت. ومع ذلك ، يمكن ببساطة جعل

يقوم TBrowse بعرض البيانات الجديدة على الشاشة بواسطة الطريقتين `b:refreshAll()` و `b:refreshCurrent()` (التي سيتم بحثهما بتفصيل أكثر لاحقاً).
تحدد هاتان الطريقتان كافة صفوف البيانات أو صفاً واحداً فقط على أنه غير صحيح ،
مما يؤدي إلى قيام الوظيفة `b:stabilize()` بإعادة عرضها. إن وجود كيائين منفصلين
يعني أيضاً أن هناك مؤشرين منفصلين يتم التعامل معهما: (أ) مؤشر سجل في قاعدة
البيانات الخاصة بك ، و (ب) مؤشر الصف في البيانات المعروضة في نافذة TBrowse.
ويتيح المتغير الفوري `b:skipBlock` لك القيام بأعمال مسلية بتغييرك هذه
المؤشرات. وسنبحث المتغير `b:skipBlock` بتفصيل أكثر في فقرات تالية.

معالجة الضغط على المفاتيح العامة

نفترض أن المفاتيح ذاتها ستستخدم في معظم أمثلة TBrowse التي سنعرضها.
وللاختصار سننشئ واحدة تقبل متغيرين اثنين: هدف TBrowse وضغطة مفتاح.
ويتم اختبار وضغطة المفتاح والعمل عليها ضمن سياق هدف TBrowse ، وقبل كتابة
هذه الوظيفة سنلقي نظرة على بقية طرق التحرك خلال TBrowse:

الوظيفة	العرض
<code>down()</code>	التحرك إلى أسفل سطر واحد
<code>end()</code>	التحرك إلى أقصى اليمين للعمود المرئي
<code>goBottom()</code>	التحرك إلى نهاية الملف
<code>goTop()</code>	التحرك إلى أعلى الملف
<code>home()</code>	التحرك إلى أقصى اليسار لعمود البيانات المرئي
<code>left()</code>	التحرك إلى اليسار بمقدار عمود واحد
<code>pageDown()</code>	التحرك إلى الشاشة التالية المملئة بالبيانات
<code>pageUp()</code>	التحرك إلى الشاشة السابقة المملئة بالبيانات
<code>panEnd()</code>	التحرك إلى أقصى اليسار للعمود
<code>panHome()</code>	التحرك إلى أقصى اليمين للعمود المرئي

الجدول مستمر من الصفحة السابقة....

الوظيفة	(تابع) العرض
panLeft()	الدوران إلى اليسار بدون تغيير موقع المؤشر
panRight()	الدوران إلى اليمين بدون تغيير موقع المؤشر
right()	التحرك إلى اليمين بمقدار عمود واحد
up()	التحرك إلى الأعلى بمقدار سطر واحد

وفيما يلي الوظيفة التي سنستخدمها في العديد من أمثلة TBrowse اللاحقة:

```
#include "inkey.ch"

#define ALREADY_PROCESSED 123456

function keytest(nKey, oBrowse, aPrekeys_, aPostkeys_)
local lProcessed := .t.
local nEle
if aPrekeys_ <> NIL .and. ;
    ( nEle := ascan(aPrekeys_, { | a | nKey == a[1] } ) ) > 0
    nKey := eval(aPrekeys_[nEle, 2], oBrowse)
endif
if nKey <> NIL
do case
case nKey == K_UP
oBrowse:up()
case nKey == K_DOWN
oBrowse:down()
case nKey == K_LEFT
oBrowse:left()
case nKey == K_RIGHT
oBrowse:right()
case nKey == K_PGUP
oBrowse:pageUp()
case nKey == K_PGDN
oBrowse:pageDown()
case nKey == K_CTRL_PGUP
oBrowse:goTop()
case nKey == K_CTRL_PGDN
oBrowse:goBottom()
case nKey == K_HOME
oBrowse:home()
case nKey == K_END
```

```

oBrowse:end()
case nKey == K_CTRL_HOME
oBrowse:panHome()
case nKey == K_CTRL_END
oBrowse:panEnd()
case nKey == K_CTRL_LEFT
oBrowse:panLeft()
case nKey == K_CTRL_RIGHT
oBrowse:panRight()
otherwise
IProcessed := (nKey == ALREADY_PROCESSED)
endcase
if aPostkeys_ <> NIL .and. ;
( nEle := ascan(aPostkeys_, { | a | nKey == a[1] } ) ) > 0
nKey := eval(aPostkeys_[nEle, 2], oBrowse)
if ! IProcessed
IProcessed := (nKey == ALREADY_PROCESSED)
endif
endif
endif
return IProcessed

```

يمكن استخدام هذه الوظيفة لأي هدف TBrowse مهما كان ، وهذا يوفر علينا كتابة شيفرة المصدر الخاصة ببنية اختبار المفاتيح CASE مرة ثانية.

قد تتساءل عن المتغيرين الثالث والرابع هذين فإنهما يقومان بدور سنارة صيد قوية في هذه الوظيفة مما يتيح لنا أن: (أ) نخصص مفاتيح فعالة إضافية ، (ب) أو نلغي الإجراءات الافتراضية ، (ج) أو نزيد الحمل على الإجراءات الافتراضية. **aPrekeys_** و **aPostkeys_** ستكون مصفوفات تتضمن مصفوفات متداخلة. ويجب أن تكون بنية كل مصفوفة متداخلة على النحو التالي:

```
{ <key> , <action block> }
```

حيث <Key> هي قيمة (INKEY) الخاصة بضغطة المفاتيح. و <action block> هي كتلة الشيفرة التي سيتم تقييمها في حال ضغط <Key>. تتم معالجة المفاتيح المحددة

في الوظيفة `aPrekeys_` قبل بنية الأمر القياسي CASE ، بينما تتم معالجة المفاتيح المحددة في الوظيفة `aPostkeys_` بعد بنية الأمر القياسي CASE.

تشغيل المفاتيح الإضافية

سنقوم في المثال التالي بتشغيل مفتاحين إضافيين هما : "A" لإضافة سجل أو `[Enter]` لإدخال تعديل أو تحرير خلية ما. وتبقى جميع المفاتيح الأخرى في الوظيفة `KeyTest()` عاملة في حالة عدم الضغط على "A" أو `[Enter]`.

```
local keys_ := { { chr(65), { | | addrec( ) } } , ;  
                  { K_ENTER, { | | editrec( ) } } }  
keytest(nkey, b, keys_)
```

إلغاء الإجراءات الافتراضية

إذا أردت إلغاء وظيفة افتراضية لمفتاح ما ، تمرر هذا المفتاح كجزء من المصفوفة -- `aPrekeys_` . فعلى سبيل المثال ، سيغير المثال التالي مفتاح `[Home]` و `[End]` ليقيما باستدعاء وظيفتي `JumpUp()` و `JumpDown()` على التوالي (وسيتم بحثهما لاحقاً).

```
local keys_ := { { K_HOME, { | b | jumpup(b) } } , ;  
                  { K_END, { | b | jumpdown(b) } } }  
keytest(nkey, b, keys_)
```

لاحظ أن كتلة الرموز هذه تقبل متغيراً واحداً وترسله إلى الوظيفة. وهذا لأن وظيفة `KeyTest()` ستمرر هدف `TBrowse` كمتغير إلى كتل الشيفرة `code block` الخاصة بك عندما تقوم بتقييمها. وهذه الميزة قيمة كبيرة في هذه الحالة.

زيادة حمل الإجراءات الافتراضية

يمكنك زيادة تحميل الوظيفة الافتراضية لمفتاح ما بتضمينه تلك الوظيفة. ولنفرض مثلاً أنك تريد أن ينتقل مفتاح **Ctrl-Home** إلى السجل الأول أيضاً علاوة على انتقاله إلى الخلية الأولى (وهو الإجراء الافتراضي). لإنجاز ذلك يمكنك تمرير المصفوفة التالية:

```
local keys_ := { { K_CTRL_HOME, { | b | b:goTop( ), K_CTRL_HOME } } }
```

لاحظ أن القيمة المعادة لكتلة الشيفرة ، تكون دائماً في أقصى يمين السطر. وفي مثالنا أعلاه ، نعيد قيمة INKEY لمفتاحي **Ctrl-Home** مما يؤدي إلى معالجتها ثالية ضمن بنية CASE. وسيكون علينا في النهاية تنفيذ الوظيفة **b:goTop()** بسبب المصفوفة **aPrekeys_** ، ثم تنفيذ الوظيفة **b:panhome()** بسبب بنية CASE المشفرة في البرنامج.

تجميع وظيفة **KeyTest()** هي الأفضل بين السرعة والمرونة. وتكون السرعة بصيغة بنية CASE المشفرة في البرنامج وهي أسرع بكثير من مسح المصفوفة لإجراءات ضغط المفاتيح. ورغم ذلك ، فإننا حقيقة يمكننا تمرير المصفوفة يجعل معالجة المفاتيح مفيدة ومرونة لاستخدامها حسبما نريد.

هل هذا كل مافي الأمر؟

لقد عرفنا الآن أسس كتابة TBrowse وهي :

■ تجهيز هدف TBrowse باستخدام الوظيفة **TBrowseDB()** أو الوظيفة **TBrowseNew()**.

■ تعبئتها بأهداف TBColumn التي أنشأت باستخدام الوظيفة **TBColumnNew()** ومثلت باستخدام الوظيفة **b:addColumn()**.

■ إدخال DO WHILE رئيسية تشتمل على : (أ) حلقة التوازن ، (ب) ضغطة بقية المعلومات إضافية ، والكثير منها يسهل معرفته أثناء العمل فلنبدأ الآن.

إنشاء أهداف "عمود استعراض الجداول" TBColumn

مع أن الوظيفة TBrowse عملية ، لكن ماوصلنا إليه حتى هذه المرحلة لايفيدنا كثيراً. ولحسن الحظ ، تشتمل فئة هدف TBColumn عدداً من المتغيرات الفورية التي تتحكم بما يعرضه كل عمود في هدف TBrowse. ويمكن تعديل هذه المتغيرات الفورية بحيث تصبح مكونات TBrowse أجمل بكثير مما كانت عليه.

الاسم	العرض
block	كتلة الشيفرة الخاصة باسترجاع البيانات من العمود
cargo	القائمة المعرفة من قبل المستخدم
colorblock	كتلة الشيفرة التي تحدد لون فقرات البيانات
colSe[رمز فصل العمود
defColor	مصفوفة فهارس رقمية داخل جدول الألوان
footing	تذييلة العمود
footSep	رمز فصل التذييلة
heading	ترويسة العمود
headSep	رمز فصل الترويسة
picture	فقرة الصورة (جديدة موجودة في كليب 5.2 فقط)
width	عرض عمود الاستعراض

الكتلة BLOCK

يشتمل هذا المتغير الفوري على كتلة شيفرة تحدده عند تقييمها ، البيانات التي يحتوي عليها العمود. وينبغي تمثيل هذه الكتلة كمتغير ثانٍ للوظيفة (TBColumnNew). هناك ثلاث طرق مشتركة لتعيين الكتل :

■ باستخدام الوظيفة (FIELDBLOCK) أو الوظيفة (FIELDWBLOCK)
(كما بينا سابقاً):

```
c := TBColumnNew(field(x), fieldblock(field(x)) )
c := TBColumnNew(field(x), fieldblock(field(x), select() ) )
```

■ وضع قيمة مبدئية لكتلة الشيفرة لربط عملية التجميع:

```
c := TBColumnNew("FNAME", { | | fname } )
```

■ تجميع كتلة الشيفرة أثناء عملية التشغيل:

```
mfield := "FNAME"
c := TBColumnNew(mfield, &("{ | | " + mfield + "}") )
```

تشمل الفقرات التي يمكن أن تشير إليها كتلة c:block ، حقل قاعدة البيانات ، وتعبيرات كليبر ، ومتغير ذاكرة ، وعنصر مصفوفة ، وسطر في حقل ذاكرة. بل إن كل مايمكن وضعه في كتلة شيفرة يمكن إضافته في هدف TBColumn. فيمكنك ، على سبيل المثال ، عرض رقم سجل كجزء من الاستعراض مع هذه الكتلة:

```
c := TBColumnNew(mfield, { | | recno() } )
```

كما يمكنك تخصيص عمود لعرض حالة سجلات محذوفة:

```
c := TBColumnNew(mfield, { | | if(deleted(), "<deleted>", space(9)) } )
```

سيتضمن معظم الأمثلة التالية حقول قاعدة بيانات ، وذلك لأنه لا يمكننا استعراض مصادر بيانات بديلة قبل فحص المتغير الفوري `b:skipblock` .

الشحنة Cargo

إن هذا المتغير هو حيز للبيانات يحدده مستخدم البرنامج. وسيرد لاحقاً في هذا البحث الكثير من الأمثلة على استخداماته.

كتلة الألوان ColorBlock

وصلنا إلى مرحلة استعراض TBrowse. يحتوي هذا المتغير الفوري كتلة شيفرة تحدد اللون الذي سيستخدم في عرض البيانات.

سيكون لكل هدف TBrowse متغير فوري هو جدول الألوان `colorSpec` الذي يشتمل على مجموعة من الألوان. وستحمل هذه المجموعة ، افتراضاً ، بالقيم الحالية "لضبط الألوان" (`SETCOLOR`) ، ولكن يمكنك ضبطها حسبما تريد وبسهولة. ويقوم `b:colorSpec` بدور لوحة الألوان المشتملة على كافة الألوان المتوفرة.

كقيمة افتراضية ، سيستخدم الاستعراض TBrowse اللون الثاني في `b:colorSpec` لعرض البيانات في الموقع الحالي للمؤشر. أما بقية الألوان فستعرض باللون الأول. وقد تم ضبط هذه القيم الافتراضية بواسطة المتغير الفوري "تحديد الألوان" `c:defcolor` الذي سيشرح لاحقاً. ولكن يمكنك إلغاء هذه القيم الافتراضية للبيانات التي في العمود باستخدام `c:colorBlock` الذي يفوق `c:defcolor` "تحديد الألوان" فيما يتعلق بالبيانات المتغيرة (أي البيانات غير الخاصة بالترويسة والتذييلة والفواصل).

هناك قاعدتان بسيطتان ينبغي اتباعهما عند إنشاء كتلة الشيفرة c:colorBlock :

- يجب أن يقبل c:colorBlock متغيراً مستقلاً يمثل البيانات في الخلية الحالية. وسيمرر آلياً إلى كتلة الشيفرة عن طريق TBrowse .
- يجب أن يعيد c:colorBlock مصفوفاً مكوناً من رقمين. ويكون هذان الرقمان كمؤشرين في جدول الألوان b:colorSpec ويمثلان زوجاً قياسياً للألوان التي ستستخدم لعرض تلك البيانات. والمثال التالي يوضح ذلك أكثر:

```
b:colorSpec := 'W/B, +W/N, W/R, +GR/R'
c:colorBlock := { | x | if(valtype(x) == "D".and. x == date( ) , ;
```

فإذا كان عنصر البيانات تاريخ يوافق التاريخ في النظام ، فستحتوي المصفوفة الراجعة من كتلة الألوان c:colorBlock الأعداد ٣ و ٤ ، وهذا يجعل الاستعراض TBrowse يستخدم اللونين الثالث والرابع عند عرض البيانات ، وبالتالي ستعرض باللون الأصفر فوق خلفية حمراء إذا ما ظللت ، وباللون الأبيض فوق خلفية حمراء إن لم تظل. وستعرض بقية أنواع البيانات جميعها باستخدام اللونين ١ و ٢ (الأبيض الناصع فوق الخلفية السوداء إذا ما ظللت ، والأبيض فوق الخلفية الزرقاء إن لم تظل).

تستخدم في المثال التالي خلفية أرجوانية لعرض التواريخ الموافقة للتاريخ في النظام ، وخلفية حمراء لعرض الأرقام السلبية. ويمكن في الحقيقة استخدام كتلة الألوان c:colorBlock حسب الذوق والرغبة بلا حدود.

```
#include "inkey.ch"

function tbrow03
local x
local y
local oBrowse := TBrowseDB(0, 0, 3, maxcol())
local oColumn
local nKey
local nOldcursor := setcursor(0)
local aMorekeys := { ;
```



```

        { K_F10, { | b | resetcolor(b) } } ;
    }
local aStuff := { { "Greg", ctod('05/07/61'), 0 } , ;
                  { "Barb", date(), 2 } , ;
                  { "Joe", ctod('04/01/59'), 500 } , ;
                  { "Adam", ctod('03/18/65'), -200 } }

dbcreate('tbrow03', { { "NAME", "C", 6, 0 }, ;
                     { "BIRTHDATE", "D", 8, 0 } , ;
                     { "AMOUNT", "N", 7, 2 } } )
use tbrow03 new
for x := 1 to 4
    append blank
    for y := 1 to 3
        fieldput(y, aStuff[x][y])
    next
next
scroll()
go top
// Color Number:      1      2      3      4      5      6      7      8
oBrowse:colorSpec := 'W/B, N/BG, W/R, +GR/N, W/RB, +GR/B, +W/G, +GR/G'
oColumn := TBColumnNew(, { || tbrow03->name } )
oColumn:defColor := { 1, 2 }
oColumn:colorBlock := { | x | if("Adam" $ x, { 7, 8 }, { 1, 2 }) }
oBrowse:AddColumn(oColumn)
oColumn := TBColumnNew(, { || tbrow03->birthdate } )
oColumn:defColor := { 7, 8 }
oColumn:colorBlock := { | x | if(x == date(), { 5, 6 }, { 1, 2 }) }
oBrowse:AddColumn(oColumn)
oColumn := TBColumnNew(, { || tbrow03->amount } )
oColumn:defColor := { 5, 6 }
oColumn:colorBlock := { | x | if(x < 0, { 3, 4 }, { 1, 2 }) }
oBrowse:AddColumn(oColumn)
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
    if nKey == 0
        nKey := inkey(0)
    endif
    keytest(nKey, oBrowse, aMorekeys)
enddo
use
ferase('tbrow03.dbf')
setcursor(nOldcursor)
return nil

```

static function resetcolor(oBrowse)

```

local oColumn := oBrowse:getColumn(oBrowse:colPos)
oColumn:colorBlock := makeblock(oColumn)
oBrowse:refreshAll()
return nil

```

```

static function makeblock(oColumn)
return { || oColumn:defColor }

```

فائدة

إذا أردت اختبار نوع البيانات ضمن كتلة الألوان c:colorBlock الخاص بك استخدم (VALTYPE) بدلاً من (TYPE) وذلك لأن البيانات الممرة إلى كتلة الشيفرة ستكون في نطاق محلي LOCAL ولا يمكن للوظيفة (TYPE) معرفة ما يحدث في نطاق المتغير LOCAL.

إيقاف عمل كتلة الألوان colorBlock

قد تحتاج في بعض الحالات لإيقاف عمل منطقية ألوان خاصة في نطاق c:colorBlock. وهذه العملية ليست سهلة بقدر إعادة ضبط كتلة الألوان إلى الصفر والسبب يعود إلى أن الإصدار كليبر 5.2 يحتوي ميزة إضافية لتدقيق الأخطاء لتجنب استخدام المتغيرات الفورية للاستعراض TBrowse أو لعمود الاستعراض TBColumn لأنواع غير صحيحة من البيانات. ولهذا السبب أضفنا قاعدة منطقية في صيغة وظيفة إعادة ضبط الألوان ResetColor() ووظيفة تشكيل الكتلة MakeBlock() لتبين لك كيفية إعادة ضبط كتلة الألوان colorBlock بشكل فعال. ولأنه لا بد أن يكون لدينا كتلة شيفرة فيمكن أن نجعل كتلة الشيفرة تعيد الألوان ذاتها مثل وظيفة تحديد الألوان c:defcolor. (يستخدم تشكيل الكتلة MakeBlock() مبدأ "نطاق محلي مستقل" لتشفير العلامة الخاصة بهدف العمود الحالي في البرنامج.

تجاهل الخلية الحالية

تبين لنا مما ذكر أعلاه أن وظيفة كتلة الألوان colorBlock تستخدم بشكل عام لتحديد الألوان استناداً إلى البيانات التي في الخلية. ومع ذلك ، يمكن استخدامها في حال تجاهل البيانات في الخلية الحالية. (وليس من الضروري استخدام البيانات). وقد تطرأ ظروف مخفية مثل حقل معين بقاعدة البيانات:

```
c:colorBlock := { | | if("Sulaiman" $ vendors->name, {3, 4}, {1, 2} ) }
```

توضح شيفرة المصدر التالية هذه الحالة بتظليل سجلات مختلفة حسب اسم المؤلف Sulaiman. حيث تشتمل وظيفة " من المؤلف؟ " (WhoWroteIt) على "جدول بحث" يعيد مصفوفة مؤشرات الألوان حسب اسم المؤلف.

```
#include "inkey.ch"
```

```
function tbrow03a
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local x
local cField
local oColumn
local nKey
local nFields
local nOldcursor := setcursor(0)
local cOldscreen := savescreen(0, 0, maxrow(), maxcol())
local bColorblock := { || WhoWroteIt(trim(articles->name)) }
scroll()
use articles new
oBrowse:colorSpec := 'W/N, N/W, W/R, +W/R, N/BG, +W/BG, W/RB, +W/RB,' + ;
    'W/B, +W/B, +W/G, +GR/G, R/W, B/W, W/GR, +W/GR, *GR+/N'
nFields := fcount()
for x := 1 to nFields
    cField := field(x)
    oColumn := TBColumnNew(cField, fieldblock(cField))
    oColumn:colorBlock := bColorblock
    oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
    dispbegin()
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
```

```

dispend()
if nKey == 0
    nKey := inkey(0)
endif
keytest(nKey, oBrowse)
enddo
setcursor(nOldcursor)          // restore previous cursor
restscreen(0, 0, maxrow(), maxcol(), cOldscreen)
use
return nil

```

```

static function WhoWroteIt(cName)
local aRetval := {1, 2}
do case
    case cName == "Joe Booth"
        aRetval := {3, 4}
    case cName == "Darren Forcier"
        aRetval := {5, 6}
    case cName == "Greg Lief"
        aRetval := {7, 8}
    case cName == "Ted Means"
        aRetval := {9, 10}
    case cName == "Clayton Neff"
        aRetval := {11, 12}
    case cName == "Kathy Uchman"
        aRetval := {13, 14}
    case cName == "Steve Baker"
        aRetval := {15, 16}
    case cName == "Mike Britten"
        aRetval := {17, 13}
endcase
return aRetval

```

لاحظ كيف أن استخدام الوظيفة "بداية العرض" (DISPBEGIN) والوظيفة "نهاية العرض" (DISPEND) ملفوفتان دائرياً في حلقة التثبيت. وهذا يساعد على دمج مخرجات الشاشة بحيث تظهر شاشة الاستعراض TBrowse بأكملها بدلاً من العرض العادي الذي تظهر فيه الصفوف تباعاً.

كما يوجد في الأسطوانة المرن الخاص بشيفرة المصدر المرفق مع الكتاب ملف يدعي "برنامج اللامعقول" INSANITY.PRG الذي يجعل وظيفة كتلة الألوان colorBlock مفيدة إلى أقصى حد ، بحيث يمكن دائماً تغيير ألوان الخلايا.

فاصل الأعمدة COLSEP

هي سلسلة حرفية تستخدم كفاصل للأعمدة ، فإن لم تحدد هذا الفاصل ، فإن TBrowse سيستخدم محتويات المتغير الفوري TBrowse:colSep التي هي في الحالة الافتراضية فراغ. وسيوضح المثال التالي استخدام هذا المتغير الفوري بإنشاء فواصل مختلفة لكل عمود.

```
#include "inkey.ch"

function tbrow04(cDbfname)
local aColseps := { "?", chr(176), chr(179), chr(177), chr(1), ;
                    chr(178), chr(186), chr(197) }
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
scroll()
use (cDbfname) new
for x := 1 to fcount()
    oColumn := TBColumnNew(field(x), fieldblock(field(x)))
    oColumn:colSep := chr(32) + aColseps[x % 8 + 1] + chr(32)
    oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    if nKey == 0
        nKey := inkey(0)
    endif
    keytest(nKey, oBrowse)
enddo
return nil
```

تبين لنا في المثال أعلاه أنه يمكننا استخدام أكثر من رمز واحد "لفاصل الأعمدة" C:colSep ولا حد لعدد الرموز. ولكن ينبغي من حيث المبدأ وجود عدد فردي في الرموز (ثلاثة ، بالنسبة للمثال الوارد أعلاه) بحيث يكون للفواصل الفعلي العدد ذاته من الفراغات على كلا الجانبين. وهذا يحافظ على تناسق نافذة الاستعراض.

تحديد الألوان defcolor

ورد في بحث كتلة الألوان c:colorBlock أن وظيفة "تحديد الألوان" defcolor تساعد في تحديد الألوان التي ستستخدم لعرض كل عنصر من البيانات. ويشتمل هذا المتغير الفوري على مصفوفة مكونة من رقمين يعملان كمؤشرين في المتغير الفوري "جدول الألوان" b:colorSpec. وكقيمة افتراضية ، سيحتوي c:defcolor على { 1, 2} وهذا يوضح سبب استخدام اللونين الأول والثاني في وظيفة "مواصفات الألوان" b:colorSpec.

لن نستخدم اللون الأول الذي حددته وظيفة "تحديد الألوان" defcolor لعرض بيانات غير مختارة فقط ، بل وكذلك الترويسات والتذييلات.

قد يسأل سائل: كيف يمكننا عرض ترويسة العمود بلون مختلف عن لون البيانات ؟ وقد يبدو هذا صعباً في البداية لأننا إستخدمنا نفس اللون لبيانات غير مختارة والترويسة ، والتذييلة ، والفواصل. ومع ذلك فالأمر سهل للغاية إذا استخدمنا خليطاً من المتغيرات الفورية للوظيفة "تحديد الألوان" c:defColor ووظيفة "كتلة الألوان" c:colorBlock. وبما أنه يمكن تطبيق "تحديد الألوان" c:defColor على كافة هذه العناصر ، ويمكن تطبيق "كتلة الألوان" c:colorBlock على البيانات فقط ، نضبط c:defColor بحيث نستخدم الألوان التي نريدها لعرض الترويسة والتذييلة والفواصل ثم نستخدم "كتلة الألوان" c:colorBlock لاستخدام لون مختلف لعرض البيانات.

يوضح المثال التالي هذه القاعدة المنطقية باستخدام اللون الأبيض فوق خلفية حمراء (اللون رقم ٣) لعرض الترويسة والتذييلة والفواصل.

```
#include "inkey.ch"

function tbrow05(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, 5, maxcol())
local oColumn
local nKey
local nFields
oBrowse:colorSpec := 'W/B, +GR/B, W/R'
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
  oColumn := TBColumnNew(field(x), fieldblock(field(x)))
  oColumn:defColor := {3, 3}
  oColumn:colorBlock := { || { 1, 2 } }
  oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  Keytest(nKey, oBrowse)
enddo
use
return nil
```

التذييلة Footing

المتغير الفوري Footing هو السلسلة الحرفية التي ستعرض في الصف السفلي من العمود. وكقيمة افتراضية لن يستخدم أي تذييلة. ويمكنك أيضاً تضمين الفاصلة المنقوطة " ; " في التذييلة مما يجعلها تظهر في أكثر من صف واحد. ويبين المثال التالي هذا الإجراء:

```
function tbrow06(cDbfname)
local x
```

```

local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
    cField := field(x)
    oColumn := TBColumnNew(cField, fieldblock(cField))
    //----- embed semi-colon for multiple lines in heading/footing
    oColumn:footing := 'This is the;' + cField + ' field'
    oBrowse:AddColumn(oColumn)
next
do while ! oBrowse:stabilize()
enddo
inkey(0)
use
return nil

```

فاصل التذييلة FootSep

وكذلك الحال بالنسبة لـ: footSep فهو السلسلة الحرفية التي ستستخدم لفصل الأعمدة التي في الصف الذي فوق التذييلة. وإذا لم تعين فاصلاً سيستخدم الاستعراض TBrowse محتويات المتغير الفوري "فاصل التذييلة" TBrowse:footSep : وهي غير مستخدمة ، كقيمة افتراضية. وسيوضح المثال التالي كيفية استخدامها:

```

function tbrow07(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
    cField := field(x)
    oColumn := TBColumnNew(cField, fieldblock(cField))
    oColumn:colSep := chr(32) + chr(179) + chr(32)

```



```
oColumn:footSep := chr(205) + chr(207) + chr(205)
oColumn:footing := field(x)
oBrowse:AddColumn(oColumn)
next
do while ! oBrowse:stabilize()
enddo
inkey(0)
return nil
```

مثل فواصل الأعمدة (c:colSep) ، يمكن استخدام أكثر من رمز واحد في هذه السلسلة. ومع ذلك ، من الأفضل عمل فاصل التذييل c:footSep بطول فاصل العمود c:colSep ذاته.

ملاحظة هامة

سيكرر الرمز الموجود في أقصى اليمين في صف "فاصل التذييل" c:footSep على عرض حدود TBrowse (أي تحت كل عمود).

الترويسة Heading

وهي السلسلة حرفية التي ستعرض في الصف العلوي من كل عمود. ويمكن ، كما مرّ معنا سابقاً ، تحرير هذه الرموز كمتغير أول في وظيفة "عمود استعراض جديد" (TBColumnNew). وكقيمة افتراضية ، لن يستخدم أية ترويسة. ويمكنك أيضاً تضمين الفاصلة المنقوطة ";" في الترويسة مما يجعلها تظهر في أكثر من صف واحد.

يبين المثال التالي كيفية تغيير المتغير الفوري مباشرة بدلاً من تحرير الترويسة كمتغير.

```
function tbrow08(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
```

```

for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew( , fieldblock(cField))
  oColumn:heading := cField
  oBrowse:AddColumn(oColumn)
next
do while ! oBrowse:stabilize()
enddo
inkey(0)
return nil

```

فواصل الترويسة headSep

أنها ، مثل فواصل التذييلة c:footSep ، سلسلة حرفية تستخدم لفصل الأعمدة الموجودة في الصف الذي أسفل الترويسة. وإن لم تعين فاصل ترويسة ، سيستخدم الاستعراض TBrowse محتويات المتغير القسري "فاصل الترويسة" TBrowse:headSep (وهي غير مستخدمة ، كقيمة افتراضية). وسيوضح المثال التالي كيفية استخدامها.

```

function tbrow09(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(cField, fieldblock(cField))
  oColumn:headSep := chr(205) + chr(209) + chr(205)
  oColumn:colSep := chr(32) + chr(179) + chr(32)
  oColumn:footSep := chr(205) + chr(207) + chr(205)
  oColumn:footing := cField
  oBrowse:AddColumn(oColumn)
next
do while ! oBrowse:stabilize()
enddo
inkey(0)
return nil

```

وكما هو الحال بالنسبة للمتغيرات الفورية لفواصل الأعمدة c:colSep وفواصل التذييلة c:footSep ، يمكن استخدام أكثر من رمز واحد لفواصل الترويسة c:headSep . ومع ذلك ، من الأفضل عمل فاصل الترويسة c:headSep بطول العمود c:colSep وفواصل التذييلة c:footSep ذاته ، لاحظ أن رمز الذي في أقصى اليمين لصف "فاصل الترويسة" c:headSep سيكرر تحت كل عمود من أعمدة TBrowse .

الصورة Picture (أضيف في الإصدار 5.2)

يحتوي هذا المتغير الفوري على فقرة "صورة" PICTUER للعمود وسيستخدم هذا المتغير من قبل الاستعراض TBrowse عند عرض البيانات التي في ذلك العمود.

قبل الإصدار 5.2 من كليبر ، كانت تستلزم محاكاة عبارة صورة "PICTUER" القيام بالعديد من الإجراءات باستخدام وظيفة "التحويل" TRANSFORM() . لذلك كان من الأفضل إضافة هذا المتغير الفوري. ومع ذلك ، إذا لم تكن تستخدم الإصدار 5.2 من كليبر فإن أسطوانة شيفرة المصدر المرفقة بهذا الكتاب يحتوي الإجراءات المذكورة أعلاه في الملف TBROW29.PRG .

يوضح المثال التالي استخدام المتغير الفوري "الصورة" وسيعرض ، كقيمة افتراضية كافة الحقول في قاعدة البيانات ، وذلك باستخدام "@" كقناع (بديل) لبيانات الرموز و "Y/N" وكقناع للبيانات المنطقية. وإذا أردنا التحكم أكثر (ومن منا لا يريد ذلك!) ، يمكننا تمرير مصفوفة تحتوي معلومات الحقل والصورة كمتغير ثانٍ. وفي هذه الحالة ستتحكم هذه المصفوفة بالحقول التي ستعرض وأيضاً بعبارات "الصورة" PICTUER التي ستطبق على تلك الحقول. وستكون بنية هذه المصفوفة على النحو التالي:

```
{ ;  
  { fieldname 1, picture clause 1 } , ;  
  { fieldname 2, picture clause 2 } , ;  
}
```

يتيح لك هذا المثال ضغط مفتاح [F10] لحذف عبارة الصورة PICTURE من العمود الحالي. لاحظ أنه لإنجاز ذلك ينبغي تعيين سلسلة فارغة null "فارغة" وليست "الصففر" NIL للمتغير الفوري "الصورة" picture. إن ميزة فحص النوع في 5.2 gdfv ، تمنع تعيين "الصففر" NIL لمعظم المتغيرات الفورية لعمود الاستعراض TBColumn ، وهذا ينطبق على مثالنا أعلاه:

```
#include "inkey.ch"

//----- manifest constants for the AFIELDS array
#define F_NAME 1
#define F_PICTURE 2

function tbrow09a(cDbfname, aFields)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local nFields
local cField
local cType
local aMorekeys := { { K_F10, { | b | ClearPict(b) } } }
use (cDbfname) new
//----- if fields info array was not passed as parameter,
//----- display all fields in the database
if aFields == NIL
    nFields := fcount()
else
    nFields := len(aFields)
endif
scroll()
for x := 1 to nFields
    //----- if fields info array was passed, pull field and
    //----- PICTURE from it...
    if aFields <> NIL
        oColumn := TBColumnNew(aFields[x][F_NAME], ;
                                fieldblock(aFields[x][F_PICTURE]))
        oColumn:picture := aFields[x][F_PICTURE]
    else
        cField := field(x)
        oColumn := TBColumnNew(cField, fieldblock(cField))
    end if
end for
```



```
//----- default picture clauses: upper-case for character
//----- data and "Y/N" for logicals
cType := type(cField)
if cType == "C"
    oColumn:picture := "@"
elseif cType == "L"
    oColumn:picture := "Y"
endif
endif
oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    if nKey == 0
        nKey := inkey(0)
    endif
    keytest(nKey, oBrowse, aMorekeys)
enddo
use
return nil

static function clearpict(oBrowse)
oBrowse:getColumn( oBrowse:colPos ):picture := "
oBrowse:configure() // to reflect the change on-screen
return nil
```

العرض Width

هي قيمة رقمية تتحكم بمسافة عرض كل عمود. وإذا لم تحدد هذه القيمة ، فسيكون عرض العمود على مدى:

(١) طول الترويسة

(٢) طول التذييلة

(٣) طول البيانات عند التقييم الأولي للكتلة

إذا حددت عرض المتغير الفوري ، فسيتم عند الضرورة ، بث كل من الترويسات والتذييلات والبيانات عند تجاوزها هذا الحد. وسيكون عرض البيانات المعروضة طول كتلة البيانات عند التقييم الأولي. وذلك لكافة أنواع البيانات غير البيانات الحرفية. أما البيانات الحرفية فستوسع بقدر عرض العمود col:width.

تحذير

لا يفضل استخدام الوظيفة () TRIM في المتغير الفوري للكتلة c:block ، وذلك لأن العبارة المقطوعة الأولى ستكون أقصر من العبارات الأخرى ، وهذا بدوره يؤدي إلى قطع البيانات في الصفوف الأخرى التالية.

يبين المثال التالي سهولة معالجة المتغير الفوري "العرض" c:width . ويسبب الضغط على السهمين الأيمن والأيسر إلى توسيع وتقليص العمود على التوالي. ويستخدم مفتاح [Tab] للانتقال بين الأعمدة.

تحذير

لا يمكن زيادة أو إنقاص المتغير الفوري "العرض" width ما لم يتم تعيينه بوضوح. وسنوضح في مثال لاحق كيفية استنتاج عرض العمود. بحيث يمكننا أن نستخدم المتغير الفوري "عرض العمود" column:width.

```
#include "inkey.ch"
```

```
function tbrow10
```

```
local x
```

```
local y
```

```
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
```

```
local oColumn
```

```
local nKey
```

```
local aNames := { { "PAT", "WILLIAMS"}, ;
```

```
    { "MARK", "WORTHEN"}, ;
```

```
    { "MARY", "GRIES"}, ;
```

```
    { "DOUG", "RIST" } }
```

```
dbcreate('tbrow10', { { "FNAME", "C", 8, 0 }, ;  
                      { "LNAME", "C", 8, 0 } } )  
scroll()  
use tbrow10 new  
for x := 1 to 4  
    append blank  
    // demonstration of using FIELDPUT() to assign values  
    for y := 1 to 2  
        fieldput(y, aNames[x][y])  
    next  
next  
go top  
oColumn := TBColumnNew( , { || tbrow10->fname } )  
oColumn:width := 1  
oBrowse:AddColumn(oColumn)  
oColumn := TBColumnNew( , { || tbrow10->lname } )  
oColumn:width := 8  
oBrowse:AddColumn(oColumn)  
do while nKey <> K_ESC  
    do while ! oBrowse:stabilize()  
        enddo  
    nKey := inkey(0)  
    oColumn := oBrowse:getColumn(oBrowse:colPos)  
    do case  
        case nKey == K_TAB  
            oBrowse:colPos := If(oBrowse:colPos == 1, 2, 1)  
            oBrowse:refreshCurrent()  
        case nKey == K_LEFT .and. oColumn:width > 1  
            oColumn:width--  
            oBrowse:configure()  
        case nKey == K_RIGHT  
            oColumn:width++  
            oBrowse:configure()  
    endcase  
enddo  
use  
ferase('tbrow10.dbf')  
return nil
```

في أعماق أهداف استعراض الجداول TBrowse Objects

لقد بحثنا في كيفية معالجة المتغيرات الفورية لأهداف عمود استعراض الجداول TBColumn للتحكم بإجراء ومظهر العرض لـ: TBrowse ، وهذا غيض من فيض.

فإن لأهداف الاستعراض TBrowse Objects عدداً من المتغيرات الفورية الخاصة بها مايقارب ضعف ما لأهداف عمود الاستعراض TBColumn. كما أن لها طرقاً تتيح لك التحكم أكثر بالمتغيرات الفورية.

مع أن إمكانيات الاستعراض كثيرة جداً فمن الأفضل استخدام مايزيد عن الحاجة فمزال هناك الكثير لتعلمه.

المتغيرات الفورية Instance Variables

على عكس المتغيرات الفورية الخاصة بأهداف عمود الاستعراض TBColumn ، إلا أنه لايمكن تعيين كل المتغيرات الفورية للاستعراض ، بل بعضاً منها فقط. وسنشير إليها بعلامة (" * ").

الاسم	الغرض
autolite*	قيمة منطقية للتحكم بال مؤشر المضيء
cargo*	متغير معرف من قبل المستخدم
colCount	عدد الأعمدة في إطار العرض browse
colorSpec*	جدول الألوان الخاص باستعراض TBrowse
colPos*	موقع العمود الذي عليه المؤشر الحالي
colSep*	رمز الفاصل للعمود
freeze*	التجميد (الإقفال)
gobottomblock*	كتلة الشيفرة المنفذة بـ: TBrowse:goBottom()
goTopBlock*	كتلة الشيفرة المنفذة بـ: TBrowse:goTop()
headSep*	رمز فاصل الترويسة
hitBottom*	مؤشر نهاية البيانات المتوفرة
hitTop*	مؤشر بداية البيانات المتوفرة
leftvisible	عرض العمود الأيسر
nBottom*	إحداثي استعراض الجزء السفلي من الجدول
nLeft*	إحداثي استعراض الجزء الأيسر من الجدول

الجدول مستمر من الصفحة السابقة....

الاسم	العرض
nRight*	إحداثي استعراض الجزء الأيمن من الجدول
nTop*	إحداثي استعراض الجزء الأعلى من الجدول
rightVisible	عرض الجزء الأيمن
rowCount	عدد السطور
rowPos*	موقع المؤشر على السطر (أو الصف)
skipBlock*	تجاوز كتلة
stable*	ثبات جدول العرض

التظليل الآلي autoLite (لا يمكن تعيينه)

يشتمل هذا المتغير الفوري على قيمة منطقية تشير إلى أن على هدف الاستعراض أن يظل تلقائياً البيانات التي في الموضع الحالي للمؤشر. وقيمة افتراضية ، فهو حقيقي (T.) وهذا مانريده عادةً.

ولكن قد تفضل في حالات معينة القيام بالتظليل بنفسك. وسنبين مثلاً على ذلك في بحث موضع المؤشر في الصف b:rowPos حيث سنقوم بتغيير موضع الصف في نافذة الاستعراض TBrowse يدوياً. ولإجراء التظليل يدوياً يمكن إيقاف عمل التظليل الآلي بضبطه على "غير حقيقي" (F.) ثم استخدام وظيفة التظليل (b:hilite) و وظيفة إيقاف عمل التظليل (b:deHilite) ، لتشغيل التظليل أو إيقافه على التوالي. (وسنبحث هاتين الوظيفتين لاحقاً).

الشحنة cargo (يمكن تعيينه)

إن هذا المتغير الفوري هو حيز للبيانات يحدده مستخدم البرنامج ، كما هو الحال في أهداف عمود الاستعراض TBColumn. وسيتبين لنا في أمثلة لاحقة أنه يُخدم الكثير من الأغراض المتنوعة.

عدد الأعمدة colCount

هذا المتغير الفوري قيمة رقمية تشير إلى العدد الإجمالي لأعمدة البيانات في هدف الاستعراض TBrowse object. ويبين الجزء التالي في الشيفرة كيفية استخدامه.

```
function tbrow11(dbf_file)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
scroll()
use (dbf_file) new
for x := 1 to fcount()
  oColumn := TBColumnNew(field(x), fieldblock(field(x)))
  oBrowse:AddColumn(oColumn)
  ? oBrowse:colCount // same as X loop counter
next
return nil
```

جدول الألوان ColorSpec (يمكن تعيينه)

ورد آنفاً أن هذا المتغير الفوري هو المرحلة الأولى لتحديد الألوان التي ستعرض بها البيانات. وإذا لم تحدد قيمة لجدول الألوان b:colorSpec فستُنسخ القيم الحالية لوظيفة ضبط الألوان (SETCOLOR) في هذه المصفوفة. ومع أن هذه الألوان ستعامل لاحقاً على أنها مصفوفة ، إلا أنه ينبغي إعداد جدول الألوان b:colorspec

على شكل سلسلة حرفية. ولاحذ لعدد الألوان التي يمكن تعيينها في جدول الألوان b:colorSpec. فقد استخدمنا هذه الطريقة لعدد كبير من الألوان.

سيستخدم الهدف TBrowse ، كقيمة افتراضية ، اللون الثاني لعرض البيانات في الموضع الحالي للمؤشر. أما البيانات الأخرى فستعرض جميعها باستخدام اللون الأول وهذه افتراضات تم ضبطها بواسطة المتغير الفوري "تحديد الألوان" c:defColor ، ومع ذلك يمكن إلغاؤها بربط كتلة ألوان colorBlock بكل هدف عمود استعراض TBColumn. (ولنزيد من الأمثلة المفصلة إرجع إلى الفقرات السابقة الخاصة بالمتغيرات الفورية "لكتلة الألوان" c:colorBlock و "تحديد الألوان" c:defColor).

يوضح الجزء التالي من الشيفرة كيفية تعيين "جدوال الألوان" b:colorSpec ومن وظيفة "ضبط الألوان" () SETCOLOR.

```
function tbrow12
test1()
test2()
return nil
function test1
local b := TBrowseDB(0, 0, maxrow(), maxcol())
/* already assigned from SETCOLOR() */
? b:colorSpec // "W/N,N/W,N/N,N/N,N/W"
/* change it now */
b:colorSpec := 'W/B,+W/B,W/R,+GR/R'
? b:colorSpec // "W/B,+W/B,W/R,+GR/R"
return nil

function test2
local b
/* note: change setcolor() before creating TBrowse object */
setcolor('w/b, +w/b, w/r, +gr/r, +w/rb')
b := TBrowseDB(0, 0, maxrow(), maxcol())
? b:colorSpec // "W/B,+W/B,W/R,+GR/R,+W/RB"
return nil
```

موضع المؤشر في العمود colPos (يمكن تعيينه)

يعيد هذا المتغير الفوري رقم العمود الذي يكون فيه المؤشر. ويمكن عند الحاجة إعادة تعيين قيمته.

نستخدم في المثال التالي "موضع المؤشر في العمود" b:colPos لبيان العمود الموجود فيه المؤشر. وكذلك المحتويات التالية للخلية المظلمة. كما نستخدم أيضاً المتغيرات الفورية للرؤيسة c:heading وعدد الأعمدة b:colcount و وظيفة "الحصول على عمود" b:getColumn() التي تعيد هدف عمود.

ستستخدم أيضاً المتغير الفوري "شحنة عمود الاستعراض" TBColumn:cargo لتخزين كتلة شيفرة العمود الأول فقط. وستقوم الوظيفة "عرض المعلومات" () ShowInfo بتقييم كتلة الشيفرة هذه إن كشفت. إن المتغير الفوري "الشحنة" cargo مثالية جداً لربط المعلومات الإضافية لأي عمود من الأعمدة.

```
#include "inkey.ch"
```

```
function tbrow13(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow() - 2, maxcol())
local oColumn
local nKey
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(cField, fieldblock(cField))
  //----- store cargo block for the first column only... see below
  if x == 1
    oColumn:cargo := { || setpos(maxrow() - 2, 0), ;
                      dispout("*** First Column **", '+w/b') }
  endif
  oBrowse:AddColumn(oColumn)
next
```

```
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
    if nKey == 0
      showinfo(oBrowse:getColumn(oBrowse:colPos))
      nKey := inkey(0)
    endif
    keytest(nKey, oBrowse)
  enddo
use
return nil
```

```
static function showinfo(oColumn)
local nOldrow := row()
local nOldcol := col()
local nMaxrow := maxrow()
scroll(nMaxrow - 2, 0)
// evaluate cargo block if there was one stored in this column
if valtype(oColumn:cargo) == "B"
  eval(oColumn:cargo)
endif
@ nMaxrow-1, 0 say padr("This is the " + oColumn:heading + ;
  " field", maxcol())
@ nMaxrow, 0 say "Contents: "
dispout( eval( oColumn:block ))
setpos(nOldrow, nOldcol) // restore prior cursor position
return nil
```

فاصل الأعمدة colSep (يمكن تعيينه)

كما هو الحال في أهداف عمود الاستعراض TBColumn. أن هذا المتغير الفوري هو سلسلة حرفية تستخدم كفاصل للأعمدة. فإذا لم تعينه لعمود ما فاصلاً ، سيستخدم الاستعراض TBrowse محتويات المتغير الفوري هذا. وإن لم تعين قيمة لـ:"فاصل الأعمدة" b:colSep فسيحتوي مسافة فارغة.

ينشئ المثال التالي "فاصل الأعمدة" b:colSep كما يبين كيفية إلغائه باستخدام المتغير الفوري "فاصل الأعمدة" c:colSep.

```
#include "inkey.ch"
function tbrow14(cDbfname)
local x
```

```

local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local nFields
local cField
scroll()
use (cDbfname) new
oBrowse:colSep := chr(32) + chr(179) + chr(32)
nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(cField, fieldblock(cField))
  // override oBrowse:colSep default between 1st & 2nd columns only
  if x == 2
    oColumn:colSep := chr(32) + chr(186) + chr(32)
  endif
  oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  keytest(nKey, oBrowse)
enddo
return nil

```

وكما هو الحال في المتغير الفوري c:colSep فالت غير مقيد بسلسلة حرفية واحدة لـ:
b:colSep.

فاصل التذييلة footSep (يمكن تعيينه)

كما هو الحال في أهداف عمود الاستعراض TBColumn ، فهو سلسلة حرفية ، والتي تستخدم لفصل الأعمدة التي في الصف الذي فوق التذييلة. وإذا لم تعين فاصل تذييلة لعمود ما ، سيستخدم الاستعراض TBrowse محتويات هذا المتغير الفوري وإذا لم تعين قيمة لـ "فاصل التذييلة" b:footSep فلن يستخدم.

ينشئ المثال التالي "فاصل التذييلة b:footSep كما يبين كيفية إلغائها باستخدام مرادفات العمود (c:colSep و c:footSep).

```
#include "inkey.ch"

function tbrow15(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local cField
local nFields
scroll()
use (cDbfname) new
oBrowse:colSep := chr(32) + chr(179) + chr(32)
oBrowse:footSep := chr(205) + chr(207) + chr(205)
nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(cField, fieldblock(cField))
  // override defaults between 1st and 2nd columns only
  if x == 2
    oColumn:colSep := chr(32) + chr(186) + chr(32)
    oColumn:footSep := chr(205) + chr(202) + chr(205)
  endif
  oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  Keytest(nKey, oBrowse)
enddo
return nil
```

الإقفال freeze (يمكن تعيينه)

هذه ميزة أخرى يتميز بها الاستعراض TBrowse عن وظيفة (DBEDIT). حيث يتيح لك هذا المتغير الفوري "الإقفال" b:freeze إقفال عمود أو أكثر في الجانب

الأيسر من الاستعراض TBrowser ، وهذا يصعب إنجازه من خلال وظيفة DBEDIT() ، إن لم يكن مستحيلاً. وكقيمة افتراضية ، لن يقفل أي عمود.

تحذير

لا يمكن تعيين "الإقفال" b:freeze ما لم يوجد عمود واحد على الأقل في هدف الاستعراض TBrowser الذي تعمل عليه. وإذا حاولت ذلك سيتجاهل الاستعراض هذا التعيين وستبقى القيمة الافتراضية للمتغير الفوري b:freeze صفراً.

سنقفّل في المثال التالي أقصى عمود في الجهة اليسرى: الذي يحتوي أول حقل في قاعدة البيانات. ويبين هذا المثال كيفية منع المؤشر من الدخول إلى الحقل المقفل بمعالجة المتغير الفوري "موضع المؤشر في العمود" b:colPos.

```
#include "inkey.ch"
```

```
function tbrow16(cDbfname)
local x
local oBrowse := TBrowserDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(cField, fieldblock(cField))
  oBrowse:AddColumn(oColumn)
next
oBrowse:freeze := 1 // freeze the leftmost column
do while nKey <> K_ESC
  // do not allow cursor to move into frozen columns
  if oBrowse:colPos <= oBrowse:freeze
    oBrowse:colPos := oBrowse:freeze + 1
  endif
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    nKey := inkey(0)
```

```
endif
keytest(nKey, oBrowse)
enddo
use
return nil
```

كتلة الانتقال إلى الأعلى وإلى الأسفل (يمكن تعيينها)

أعتقد أن الوقت المناسب قد حان لمناقشة الفرق بين الوظيفة (TBrowseDB) والوظيفة (TBrowseNew). في الحقيقة يرتبط بكل هدف استعراض TBrowse ثلاث كتل شيفرة تتحكم بالحركة ، وهي كتلة شيفرة الانتقال إلى الأسفل b:goBottomBlock وكتلة شيفرة الانتقال إلى الأعلى b:goTopBlock وكتلة شيفرة التجاوز b:skipBlock. عندما تنشئ الوظيفة (TBrowseDB) هدف الاستعراض TBrowse فإنها بذلك تقوم بتزويد كتل الشيفرة بافتراضات الحركة هذه. أما وظيفة (TBrowseNew) فلا توفر كتل الشيفرة هذه ، ولا يمكن لهذا السبب استخدامها إلى أن تقوم بإنشائها.

يتم تقييم وظيفة "كتلة الانتقال إلى الأسفل" (b:goBottomBlock) ووظيفة "كتلة الانتقال إلى الأعلى" (b:goTopBlock) بواسطة وظيفة "كتلة الانتقال إلى الأسفل" (goBottom()) ووظيفة "الانتقال إلى الأعلى" (goTop()) على التوالي. سنجيز في الأمثلة التالية كتلاً خاصة تقوم بالقفز إلى السجلين رقم ١٠ ورقم ٥٠ عندما نحاول الانتقال إلى أعلى أو أسفل الملف على التوالي. وما زال بإمكاننا الانتقال مروراً بالسجلات الأخرى باستخدام مفتاح السهم إلى الأعلى ومفتاح السهم إلى الأسفل ، لكننا سنبين كيفية اختصار هذا الإجراء عند بحث المتغير الفوري "تجاوز الكتلة" b:skipBlock .

```
#include "inkey.ch"
```

```
function tbrow17
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
```

```

local nKey
scroll()
dbcreate('tbrow17', { { "TOPSECRET", "C", 15, 0 } } )
use tbrow17 new
for x := 1 to 99
    append blank
    tbrow17->topsecret := "Dummy Record " + ltrim(str(x))
next
go top
oBrowse:goTopBlock := { || dbgoto(10) }
oBrowse:goBottomBlock := { || dbgoto(50) }
oColumn := TBColumnNew(, { || recno() } )
oBrowse:AddColumn(oColumn)
oColumn := TBColumnNew(, { || tbrow17->topsecret } )
oBrowse:AddColumn(oColumn)
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    if nKey == 0
        nKey := inkey(0)
    endif
    keytest(nKey, oBrowse)
enddo
use
ferase('tbrow17.dbf')
return nil

```

فواصل الترويسة headSep (يمكن تعيينه)

كما هو الحال في أهداف عمود الاستعراض TBColumn ، فهي سلسلة حرفية ، والتي تستخدم لفصل الأعمدة التي في الصف الذي أسفل الترويسة ، وإذا لم تعين فاصل ترويسة لعمود ما ، سيستخدم الاستعراض TBrowse محتويات المتغير الفوري هذا. وإذا لم تعين قيمة لـ: "فاصل الترويسة" b:headSep فلن يُستخدم.

ينشئ المثال التالي متغيرات جميع الفواصل الثلاثة (فاصل الأعمدة b:colSep و فاصل التذييل b:footSep و فاصل الترويسة b:headSep) كما يوضح كيفية إلغائها بالمتغيرات الفورية للأعمدة.

```
#include "inkey.ch"

function tbrow18(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local nFields
local cField
scroll()
use (cDbfname) new
oBrowse:colSep := chr(32) + chr(179) + chr(32)
oBrowse:headSep := chr(205) + chr(209) + chr(205)
oBrowse:footSep := chr(205) + chr(207) + chr(205)
nFields := fcount()
for x := 1 to nFields
    cField := field(x)
    oColumn := TBColumnNew(cField, fieldblock(cField))
    // override defaults between 1st & 2nd columns only
    if x == 2
        oColumn:colSep := chr(32) + chr(186) + chr(32)
        oColumn:headSep := chr(205) + chr(203) + chr(205)
        oColumn:footSep := chr(205) + chr(202) + chr(205)
    endif
    oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    if nKey == 0
        nKey := inkey(0)
    endif
    keytest(nKey, oBrowse)
enddo
use
return nil
```

وكما هو الحال في "فواصل الترويسة" c:headSep يمكنك استخدام أكثر من رمز واحد لفواصل الترويسة b:headSep وإذا استخدمنا رمزاً واحداً فقط فسيكرر على عرض حدود الاستعراض TBrowse.

تجاوز الحد الأسفل والحد الأعلى

إن تجاوز الحد الأسفل hitBottom وتجاوز الحد الأعلى hitTop (يمكن تعيينهما).

إن هذين المتغيرين الفوريين قيمتان منطقيتان تشيران إلى أن هناك محاولة لتجاوز نهاية أو بداية البيانات المتوفرة ، وقيمتها العاديتان "غير حقيقي" (.F.) وفي حال محاولة تجاوز أسفل أو أعلى البيانات فستضبط قيمة b:hitBottom أو b:hitTop على "حقيقي" (.T.) ، على التوالي. ويظهر ذلك أثناء عملية تثبيت stibilization وفي حال عدم تمكن "تجاوز الكتلة" b:skipBlock من تجاوز عدد السجلات المطلوبة إلى الأمام أو إلى الخلف.

لاحظ أنه ليس من الضرورة أن تساوي هذه المتغيرات الوظائف BOF() و EOF(). فإنها تنطبق على كل ما يراه الاستعراض TBrowse كحد للبيانات الصحيحة المتوفرة. وإذا عدلنا كتل الحركة الثلاث فمن المحتمل جداً الاصطدام بأعلى البيانات وأسفلها المتوفرة دون الحاجة لتغيير قيمة BOF() و EOF().

نستخدم في المثال التالي هذين المتغيرين الفوريين لتنبيه المستخدم عند إصطدامه بأعلى أو أسفل البيانات.

```
#include "inkey.ch"
```

```
function tbrow19(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow() - 1, maxcol())
local oColumn
local nKey
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
    cField := field(x)
    oColumn := TBColumnNew(cField, fieldblock(cField))
    oBrowse:AddColumn(oColumn)
```

```
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
    if nKey == 0
      nKey := inkey(0)
    endif
    if nKey == K_UP
      oBrowse:up()
      oBrowse:stabilize()
      if oBrowse:hitTop
        alert("Top of data")
      endif
    elseif nKey == K_DOWN
      oBrowse:down()
      oBrowse:stabilize()
      if oBrowse:hitBottom
        alert("Bottom of data")
      endif
    else
      keytest(nKey, oBrowse)
    endif
  enddo
enddo
use
return nil
```

عرض العمود الأيسر leftVisible وعرض العمود الأيمن rightVisible

يتعقب هذان المتغيران الفوريات الأعمدة (غير المقفلة) في أقصى اليسار وأقصى اليمين في نافذة العرض TBrowse. ويسهل وجودهما عملية عرض الأسهم التي تشير إلى أن هناك بيانات غير معروضة على الشاشة.

يوضح المثال التالي استخدام هذين المتغيرين الفوريين. وقد أضفنا على الجانب الأيمن من الشاشة مسطرة تحريك عمودية تعمل مع فهرس التحكم باستخدام وظيفة NTXPOS(). تحدد هذه الوظيفة الموضع التناسبي لسجل ما ، ضمن فهرس التحكم. وللقيام بذلك ينبغي ربط الملفين NTXPOS.OBJ و NTXHAND.OBJ

(الموجودين في أسطوانة شيفرة المصدر الموجود معك) برنامجك التطبيقي لكي يعمل
هذان المتغيران.

```
#include "box.ch"
#include "inkey.ch"

#define ARROW '+w/r'

function tbrow20(cDbfname, cNtxname)
local x
local oBrowse := TBrowseDB(1, 1, maxrow() - 1, maxcol() - 1)
local oColumn
local nKey
local nFields
local cField
local nStatusRow
local oldcursor := setcursor(0)
scroll()
@ oBrowse:nTop - 1, oBrowse:nLeft - 1, ;
  oBrowse:nBottom + 1, oBrowse:nRight + 1 box B_DOUBLE + ''

//----- draw elevator bar on right side of box for vertical scrollbar
@ oBrowse:nTop, oBrowse:nRight + 1, ;
  oBrowse:nBottom, oBrowse:nRight + 1 box replicate(chr(176), 9)
nStatusRow := oBrowse:nTop + 1

use (cDbfname)

//----- set index if it was passed as a parameter
if cNtxname <> NIL
  dbsetindex(cNtxname)
endif

nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(cField, fieldblock(cField))
  oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    showarrows(oBrowse, @nStatusRow)
    nKey := inkey(0)
```

```
endif
keytest(nKey, oBrowse)
enddo
use
setcursor(oldcursor)
return nil
```

```
static function showarrows(oBrowse, nStatusRow)
local nEle
local nOldRow := row()
local nOldCol := col()
if oBrowse:leftvisible - oBrowse:freeze > 1
    @ oBrowse:nBottom + 1, oBrowse:nLeft say ;
        chr(17) + chr(196) color ARROW
else
    @ oBrowse:nBottom + 1, oBrowse:nLeft say chr(205)+chr(205)
endif
if oBrowse:rightvisible < oBrowse:colCount
    @ oBrowse:nBottom+1, oBrowse:nRight-1 say ;
        chr(196) + chr(16) color ARROW
else
    @ oBrowse:nBottom+1, oBrowse:nRight-1 say chr(205)+chr(205)
endif
```

```
//----- determine relative position
if ! empty(indexkey(0))
    nEle := ntxpos(indexord(), recno())
else
    nEle := recno()
endif
```

```
//----- determine if status row has changed
if nStatusRow <> oBrowse:nTop + int((nEle / lastrec()) * ;
    (oBrowse:nBottom - oBrowse:nTop))
```

```
dispbegin()
```

```
//----- first, blank out previous status bar
@ nStatusRow, oBrowse:nRight + 1 say chr(176)
```

```
//----- then recalculate position of status bar
nStatusRow := oBrowse:nTop + int((nEle / lastrec()) * ;
    (oBrowse:nBottom - oBrowse:nTop))
```

```
//----- finally, redraw it
@ nStatusRow, oBrowse:nRight + 1 say chr(219)

dispnd()

endif
setpos(nOldRow, nOldCol)
return nil
```

إحداثيات الاستعراض nBottom , nleft , nRight , nTop

تشتمل هذه المتغيرات الفورية الأربعة على أرقام تحدد إحداثيات نافذة بيانات الاستعراض TBrowse. ويتم تعيينها بتمريرها كمتغيرات للوظيفة (TBrowseDB) أو (TBeowseNew) ، ويمكن أيضاً معالجتها مباشرة.

يمكنك المثال التالي من تصغير أو توسيع نافذة الاستعراض TBrowse بالضغط على مفتاحي [Del] أو [Ins] على التالي.

```
#include "inkey.ch"
#include "box.ch"

static cBuffer // underlying screen -- must be visible throughout

function tbrow21(cDbfname)
local x
local oBrowse := TBrowseDB(1, 1, maxrow()-1, maxcol()-1)
local nKey
local aMorekeys := { { K_INS, { | b | grow(b) } }, ;
                     { K_DEL, { | b | shrink(b) } } }

//----- draw silly backdrop
dispbegin()
for x := 0 to maxrow()
  @ x,0 say replicate(chr(64 + x), maxcol() + 1)
next
cBuffer := savescreen(0, 0, maxrow(), maxcol())
scroll()
dispnd()
```



```

use (cDbfname) new
for x := 1 to fcount()
  oBrowse:AddColumn(TBColumnNew(field(x), fieldblock(field(x))))
next
@ oBrowse:ntop - 1, oBrowse:nleft - 1, oBrowse:nbottom + 1, oBrowse:nright + 1
box B_SINGLE + ''
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  keytest(nKey, oBrowse, aMorekeys)
enddo
use
return nil

```

```

static function shrink(oBrowse)
if oBrowse:nTop <> oBrowse:nBottom
  dispbegin()
  restscreen(0, 0, maxrow(), maxcol(), cBuffer) // restore underlying screen
  oBrowse:nTop++
  oBrowse:nBottom--
  oBrowse:nLeft++
  oBrowse:nRight--
  oBrowse:invalidate()
  @ oBrowse:ntop - 1, oBrowse:nleft - 1, ;
  oBrowse:nbottom + 1, oBrowse:nright + 1 box B_SINGLE + ''
  dispend()
endif
return nil

```

```

static function grow(oBrowse)
dispbegin()
restscreen(0, 0, maxrow(), maxcol(), cBuffer) // restore underlying screen
oBrowse:nTop--
oBrowse:nBottom++
oBrowse:nLeft--
oBrowse:nRight++
oBrowse:invalidate()
@ oBrowse:ntop - 1, oBrowse:nleft - 1, ;
oBrowse:nbottom + 1, oBrowse:nright + 1 box B_SINGLE + ''
dispend()
return nil

```

تعالج الوظائف "تقليص" (Shrink) و "زيادة" (Grow) المتغيرات الفورية الأربعة نافذة البيانات ، كما يستلزمان استخدام وظيفة "إبطال الاستعراض" TBrowse invalidate() التي تجعل هدف الاستعراض TBrowse يعيد عرض كافة الصفوف في نافذة البيانات. وهذا الإجراء ضروري؛ فبدونه لن يعاد عرض البيانات إلى أن يضغط المفتاح التالي للانتقال والمرور . لاحظ أنه لا ضرورة لاستخدام وظيفة "تجديد الجميع" refreshAll() لأن البيانات الأساسية لم تتغير. لاحظ استخدام الوظيفة "بداية العرض" (DISPBEGIN "نهاية العرض" (DISPEND لمنع اضطراب الشاشة عند استرجاع الشاشة الأساسية في وظيفتي "تقليص" (Shrink) و "زيادة" (Grow).

عداد الصفوف rowCount

إنه قيمة عددية تشير إلى إجمالي عدد صفوف البيانات المرئية في هدف الاستعراض TBrowse. ولا يشمل هذا العدد الصفوف المحتوية للترويسات ، أو التذييلات ، أو التي فيها فواصل. في المثال أدناه وسنستخدم b:rowCount في b:rowPos كما هو موضح أدناه.

موضع المؤشر في الصف rowpos

يستخدم هذا المتغير الفوري مع المتغير الفوري "موضع المؤشر في العمود" colPos وهو يحتوي رقم الصف الذي يكون عليه المؤشر.

يبدأ ترقيم الصفوف برقم (١) في أعلى الشاشة نزولاً إلى أسفلها. لاحظ أن الصفوف التي تحتوي ترويسات أو تذييلات أو فواصل لا تعتبر صفوف بيانات.

نستخدم في المثال التالي "موضع المؤشر في الصف" b:rowPos "لإقفال" صف المؤشر. أي سيكون لدينا دائماً حاجز في عدة صفوف عند الاقتراب إلى أعلى نافذة الاستعراض أو أسفلها. وسيبقى المؤشر ثابتاً بينما تتحرك البيانات تحته. إن الاستعراض

TBrowse (أو DBEDIT) يسمح عادة للمستخدم بتحريك المؤشر إلى أول أو آخر صف مرئي قبل تحريك البيانات.

```
#include "inkey.ch"
#include "box.ch"

function tbrow22
local oBrowse
local oColumn
local nKey
local nTemprow
local x
local nOldcursor := setcursor(0)
local cOldcolor := setcolor()
scroll()
//----- create temporary database
dbcreate("dummy", { { "NAME", "C", 12, 0 } })
use dummy
append blank
dummy->name := "First record"
for x := 2 to 59
    append blank
    dummy->name := "Grumpfish " + ltrim(str(x))
next
append blank
dummy->name := "Last record"
scroll()
setcolor('+w/rb')
@ 0, 33, maxrow(), 46 box B_SINGLE + ''
oBrowse := TBrowseDB(1, 34, maxrow() - 1, 45)
oBrowse:headSep := chr(205)
oColumn := TBColumnNew('Grump Number', fieldblock('NAME'))
oBrowse:AddColumn(oColumn)
oBrowse:autolite := .f.
go top
do while nKey <> K_ESC
    dispbegin()
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    dispend()
    //----- reset row position if necessary
    if nTemprow <> NIL
        oBrowse:rowPos := nTemprow
```

```

nTemprow := NIL
//----- following loop necessary to properly reset data source
do while ! oBrowse:stabilize()
enddo
endif
if nKey == 0
  oBrowse:hilite()
  nKey := inkey(0)
  oBrowse:dehilite()
endif
do case
  case nKey == K_UP
    oBrowse:up()
    //----- if we are within 3 rows of the top row, re-assign row #

    //----- unless we are within 3 records of BOF()
    if oBrowse:rowPos < 4 .and. ! AlmostBOF()
      nTemprow := oBrowse:rowPos
      oBrowse:rowpos := 1
    endif
  case nKey == K_DOWN
    oBrowse:down()
    //----- if we are within 3 rows of the bottom, re-assign row #

    //----- unless we are within 3 records of EOF()
    if oBrowse:rowPos > oBrowse:rowcount - 3 .and. ! AlmostEOF()
      nTemprow := oBrowse:rowPos
      oBrowse:rowpos := oBrowse:rowcount
    endif
  otherwise
    keytest(nKey, oBrowse)
  endcase
enddo
setcursor(nOldcursor)
setcolor(cOldcolor)
use
ferase("dummy.dbf")
return nil

```

```

/*
  Function: AlmostBOF()
  Purpose: Determine if we are within 3 records of the top-of-file
           If so, no need to limit cursor movement upwards
  Returns: .T. if near BOF(), .F. if not
*/
static function AlmostBOF

```

```
local nRec := recno()
local IRetval
skip -3
IRetval := bof()
go nRec
return IRetval
```

Function: AlmostEOF()

Purpose: Determine if we are within 3 records of the bottom-of-file

If so, no need to limit cursor movement downwards

Returns: .T. if near EOF(), .F. if not

*/

```
static function AlmostEOF
```

```
local nRec := recno()
```

```
local IRetval
```

```
skip 3
```

```
IRetval := eof()
```

```
go nRec
```

```
return IRetval
```

كتلة التجاوز Skipblock

يشتمل هذا المتغير الفوري الهام جداً على كتلة شيفرة تتحكم عند تقييمها بالحركة ضمن هدف الاستعراض TBrowse. فهو يقبل قيمة متغير مستقل واحد ، وهي قيمة رقمية يمررها الاستعراض TBrowse آلياً. وتمثل هذه القيمة الرقمية عدد السجلات المراد تجاوزها. وعلى سبيل المثال ، سيمرر السهم إلى أسفل [↓] رقم (١) والسهم إلى أعلى [↑] رقم (١-) ، ومفتاح "إلى أسفل الصفحة" [PgDn] سيمرر عدد الصفوف الموجودة في شاشة بيانات واحدة.

وكما ورد آنفاً ، تتوفر قيمة الافتراضية لكتلة التجاوز b:skipBlock عندما ننشئ هدف الاستعراض TBrowse بوظيفة "قاعدة بيانات الاستعراض" (TBrowseDB) وتعالج هذه القيمة الافتراضية كافة مفاتيح الانتقال والحركة. ومع ذلك ، يمكنك بعد الاستيعاب الكامل لاستخدامات "كتلة التجاوز" b:skipBlock

كتابة نسختك الخاصة من هذا المتغير الفوري للبلوغ بأداء مكونات الاستعراض إلى أعلى مستوى.

إن إحدى المشاكل التي تواجه مستخدمي وظيفة "التحرير في قاعدة البيانات" DBEDIT() هي سوء وبطء أدائه عند ضبط "مرشح البيانات" FILTER . فعندما نصل إلى أعلى أو أسفل نطاق البيانات ونحاول تجاوزه سيتوقف العمل ويمضي وقت لا بأس به قبل توقف وظيفة "التحرير في قاعدة البيانات" DBEDIT() عن البحث عبثاً عن السجل التالي.

يبين المثال التالي كيفية حل هذه المشكلة مع الاستعراض TBrowse. فباستخدام كتل الحركة الثلاث ينحصر الوصول إلى السجلات بتلك المؤرخة في شهر يونيو ١٩٩٣.

```
#include "inkey.ch"

function tbrow23
local x
local y
local oBrowse := TBrowseDB(2, 0, maxrow(), maxcol())
local oColumn
local nKey
local cOldDateFormat := set(_SET_DATEFORMAT, "mm/dd/yy")
local dStart := ctod('12/31/93'), dEnd := ctod('12/31/93')
scroll()
dbcreate('tbrow23', { { "DATE", "D", 8, 0 } } )
use tbrow23 new
index on tbrow23->date to tbrow23
y := ctod('12/31/92')
set(_SET_DATEFORMAT, cOldDateFormat)
for x := 1 to 365
    append blank
    tbrow23->date := x + y
next
dbseek(dStart, .t.)
// the next three statements are the heart of this example
oBrowse:goTopBlock := { || dbseek(dStart, .t.) }
oBrowse:goBottomBlock := { || dbseek(dEnd + 1, .t.), dbskip(-1) }
oBrowse:skipBlock := { | nSkipCnt | gilligan(nSkipCnt, ;
    { || tbrow23->date}, dStart, dEnd) }
oColumn := TBColumnNew("Rec #", { || recno() } )
oBrowse:AddColumn(oColumn)
```

```
oColumn := TBColumnNew(" Date", { || tbrow23->date } )
oBrowse:AddColumn(oColumn)
@ 0, 16 say 'Calendar Year 1993 - Restricted to month of June'
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  keytest(nKey, oBrowse)
enddo
use
ferase('tbrow23.dbf')
ferase('tbrow23.ntx')
return nil
static function gilligan(nSkipCnt, bVal, startval, endval)
local nMovement := 0
do case

  // no movement... flush buffers
  case nSkipCnt == 0
    skip 0

  // moving forward
  case nSkipCnt > 0
    do while nMovement < nSkipCnt .and. eval(bVal) <= endval .and. ! eof()
      skip 1
      nMovement++
    enddo
    // make sure that we are within range -- if not, move backward
    do while (eval(bVal) > endval .or. eof()) .and. ! bof()
      skip -1
      nMovement--
    enddo
    if bof()      // no data in range... fall out
      keyboard chr(K_ESC)
    endif

  // moving backward
  case nSkipCnt < 0
    do while nMovement > nSkipCnt .and. eval(bVal) >= startval
      skip -1
      if bof()
        exit
      endif
      nMovement--
    enddo
```

```
// make sure that we are within range -- if not, move forward
do while eval(bVal) < startval .and. ! eof()
  skip
  nMovement++
enddo
if eof()      // no data within range... fall out
  keyboard chr(K_ESC)
endif
endcase
return nMovement
```

لقد شكلت "الانتقال إلى كتلة أعلى" b:goTopBlock و"الانتقال إلى كتلة أسفل" b:goBottomBlock بحيث تستدعيان وظيفة "البحث في قاعدة البيانات" dbSeek() التي تقفز إلى قيمة منخفضة أو عالية في نطاق السجلات المتوفرة. لاحظ أن المتغير الثاني في وظيفة "كتلة الانتقال إلى أسفل" b:goBottomBlock تستدعي وظيفة "البحث في قاعدة البيانات" dbSeek() ، فهذا يدل على أن عملية البحث عن البيانات ستنفذ SOFTSEEK . وهذا مهم لأنه في حال عدم وجود قيمة معينة ينبغي عدم ترك مؤشر السجل عند نهاية الملف ، فقد يؤدي ذلك إلى ضياع كل شيء.

تم تشكيل المتغير الفوري "كتلة التجاوز" b:skipBlock ليقبل قيمة متغير مستقل واحد وهي قيمة رقمية يمررها الاستعراض TBrowse آلياً. ثم يستدعي وظيفة تجاوز خاصة وهي الوظيفة () Gilligan بتمرير هذه القيمة الرقمية كمتغير أول. وتشمل المتغيرات الإضافية كتلة شيفرة (تسرجع قيمة حقل قاعدة البيانات). والقيم المنخفضة والعالية لنطاق البيانات.

تقوم الوظيفة () Gilligan بواحد من ثلاث إجراءات حسب الاتجاه الذي تنتقل به في الاستعراض TBrowse:

- إذا لم يكن هناك انتقال (أي أن قيمة "عداد التجاوز" nSkipcnt يساوي الصفر) عندها تقوم الوظيفة () Gilligan بالقفز بقيمة صفر ، لمسح الذاكرة المؤقتة ، وتعيد قيمة الصفر.

■ إذا كنا ننتقل إلى الأمام ، تبدأ وظيفة () Gilligan بالدوران محاولة الانتقال إلى الأمام بعدد السجلات المطلوبة ، وتؤكد في الوقت ذاته من أن التاريخ مازال ضمن النطاق المحدد. وإذا كان مؤشر السجل ، بعد انتهاء الدوران ، خارج النطاق الصحيح فستنتقل إلى الوراء ، وهذا مطلوب لأن مؤشر السجل قد يكون في السجل الأول تحت نطاق البيانات المحدد.

■ إذا كنا ننتقل إلى الوراء ، تبدأ وظيفة () Gilligan بالدوران محاولة نقل مؤشر السجل إلى الوراء بعدد السجلات المطلوبة ، وتؤكد في الوقت ذاته بأن التاريخ مازال ضمن النطاق المحدد. وإذا كان مؤشر السجل ، بعد انتهاء الدوران ، خارج النطاق الصحيح فستنتقل إلى الأمام. وهذا أيضاً مطلوب لأنه قد يبقى مؤشر السجل في السجل الأول فوق نطاق البيانات المحدد.

لاحظ أن المتغير المحلي "رقم الانتقال" nMovement في كل حلقة من حلقات دوران الوظيفة () Gilligan يزيد أو ينقص مع انتقال مؤشر السجل ، ثم تعاد قيمته إلى هدف الاستعراض Tbrowse. وهذا هام جداً لأنه يجب أن تعاد قيمة رقمية إلى هدف الاستعراض لتبين عدد الصفوف التي ينتقل فيها الاستعراض Tbrowse فعلاً. ومع أنك لا تحتاج لذلك إذا استخدمت "كتلة التجاوز" skipBlock الافتراضية ، إلا أنه من الضروري تذكر هذا الإجراء عند استخدام المتغير الفوري "كتلة التجاوز" skipBlock الخاصة بك.

ملاحظة

ينطبق هذا المثال فقط على مفاتيح الفهرس (index keys) الخاصة بالتاريخ والأرقام. للإطلاع على مثال على كيفية ضبط كتل الانتقال مع مفتاح فهرس حرفي ، راجع موضوع: مشاهدة المجموعات الفرعية (على الطاير).

ثبات stable

يعيد هذا المتغير الفوري قيمة منطقية إذا كانت البيانات المحتملة قد عرضت على الشاشة بشكل صحيح أم لا. فإذا كان العرض بشكل صحيح ، فإنه يعيد القيمة "حقيقي" وإلا "غير حقيقي". إن هذا المتغير الفوري مرتبط ارتباطاً وثيقاً بالوظيفة "تأسيس" `b:stabilize()`.

وظائف TBrowse ذات الأغراض الخاصة

تعتبر وظائف الحركة الخاصة بالاستعراض TBrowse التي بحثناها أعلاه بسيطة إذا ما قورنت بالوظائف التي سنبحثها في هذه الفقرة ، والتي تؤدي العديد من الأغراض داخل فئة هدف TBrowse.

اسم الوظيفة	الغرض منها
<code>addColumn()</code>	إضافة الهدف TBColumn للهدف TBrowse
<code>colorRect()</code>	وظيفة تلوين المستطيل
<code>colWidth()</code>	وظيفة عرض العمود
<code>configure()</code>	وظيفة التهيئة
<code>deHilite()</code>	وظيفة إيقاف التظليل الآلي
<code>delColumn()</code>	وظيفة حذف العمود
<code>forceStable()</code>	التثبيت الجبري
<code>getColumn()</code>	استرجاع هدف عمود
<code>insColumn()</code>	إدراج عمود
<code>invalidate()</code>	وظيفة تحديث العرض
<code>hilite()</code>	تظليل الخلية الحالية
<code>refreshAll()</code>	تحديث البيانات بالكامل
<code>refreshCurrent()</code>	تحديث البيانات الحالية
<code>setColumn()</code>	وظيفة تجهيز العمود
<code>stabilize()</code>	وظيفة التثبيت

وظيفة إضافة عمود (addColmn)

تضيف هذه الوظيفة عموداً إلى هدف الاستعراض Tbrowse ، كما تزيد قيمة المتغير الفوري "عداد الأعمدة" b:colCount. وقد مر معنا آلاف العديد من الأمثلة على هذه الوظيفة.

وظيفة تلوين المستطيل (colorRect)

تمكنك هذه الوظيفة من تغيير لون مجموعة المستطيلات الخاصة بالخلايا في الاستعراض Tbrowse. وتقبل هذه الوظيفة (b:colorRect) قيمتي متغيرين مستقلين كما يلي:
(b:colorRect(<acoords> , <aColors>)

حيث أن "الإحداثيات" <aCoords> مصفوفة تحتوي أربع قيم رقمية تمثل إحداثيات المنطقة التي ستلون. ملاحظة هامة: تمثل هذه الإحداثيات مؤشرات الخلايا ، وليس إحداثيات الشاشة. فمثلاً: الأرقام { 1, 1, 2, 2 } تشير إلى المنطقة المحصورة بصف البيانات رقم (1) وهدف العمود رقم (1) ، وصف البيانات رقم (2) وهدف العمود رقم (2).

وحيث أن "الألوان" <aColors> مصفوفة تحتوي على رقمين يقومان ، مثل المتغير الفوري "تحديد اللون" c:defColor ، بدور مؤشرين في "جدول ألوان" b:colorSpec. فمثلاً: تأمر القيمة { 1, 2 } وظيفة "تلوين المستطيل" (colorRect) باستخدام اللون الثاني في "جدول الألوان" b:colorSpec لكافة البيانات التي ضمن المستطيل باستثناء موضع الخلية الذي فيه المؤشر والذي سيعرض باللون الأول.

تحفظ خلايا البيانات التي تتأثر بالوظيفة "تلوين المستطيل" (b:colorRect) بلونها الجديد عند المرور عليها أفقياً ، بينما تختفي ألوانها عند التحريك عمودياً في الشاشة.

تحذير

لا تحفظ وظيفة "تلوين المستطيل" (colorRect) ولا تستعيد موضع المؤشر. ويجب عليك كلما استخدمتها أن تقوم بنفسك بحفظ موضع المؤشر واستعادته. وسنوضح ذلك في المثال أدناه.

يبين هذا المثال كيف يمكن أن تقوم الوظيفة "تلوين المستطيل" (b:colorRect) من تظليل صف بأكمله. وسبب امتداد التظليل على كامل السطر هو أننا حددنا لفواصل الأعمدة أعمدة خاصة بها ، وإلا فلن تظلل الفواصل وسيكون التظليل على البيانات فقط.

عند الانتقال إلى صف آخر (أعلى أو أسفل) تستخدم وظيفة "تجديد الصف الحالي" (refreshCurrent) لبيان أن الصف الحالي غير صحيح ، وهذا سيجعل هدف الاستعراض TBrowse يعيد تشكيل الصف الحالي عندما نصل إلى حلقة "التأسيس" (b:stabilize) التي تلغي التظليل.

وبما أن العمود الحقيقي يكون بين عمودين "غير حقيقيين" (فواصل) فينبغي تغيير عمل مفتاحي الأسهم إلى اليسار واليمين بحيث يتحرك نقلتين إلى اليسار أو اليمين بدلاً من نقله واحدة (لتجاوز عمود فاصل الأعمدة).

```
#include "inkey.ch"
```

```
function tbrow24(cDbfname)
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local x
local nOldcursor := setcursor(0)
local cOldscreen := savescreen(0, 0, maxrow(), maxcol())
local nOldrow
local nOldcol
local nFields
scroll()
use (cDbfname) new
```

```

oBrowse:colorSpec := 'W/N,N/W,+W/R,+W/B'
oBrowse:colSep := "
nFields := fcount()
for x := 1 to nFields
    oColumn := TBColumnNew(, fieldblock(field(x)))
    oBrowse:AddColumn(oColumn)
    if x <> nFields // don't add separator after last column!
        oBrowse:AddColumn(TBColumnNew(, { || chr(32)+chr(179)+chr(32) } ) )
    endif
next
oBrowse:autoLite := .f.
do while nKey <> K_ESC
    dispbegin()
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    dispend()
    if nKey == 0
        //----- save cursor location
        nOldrow := row()
        nOldcol := col()
        //----- highlight current row
        oBrowse:colorRect( {oBrowse:rowPos, oBrowse:leftVisible, oBrowse:rowPos,
oBrowse:rightVisible}, { 3, 4 } )
        setpos(nOldrow, nOldcol)
        oBrowse:hiLite() // highlight current cell
        nKey := inkey(0)
        oBrowse:dehilite() // de-highlight current cell
    endif
    do case
        case nKey == K_UP
            oBrowse:refreshCurrent() // remove highlight from current row
            oBrowse:up()
        case nKey == K_DOWN
            oBrowse:refreshCurrent() // remove highlight from current row
            oBrowse:down()
        case nKey == K_LEFT
            oBrowse:left()
            oBrowse:left()
        case nKey == K_RIGHT
            oBrowse:right()
            oBrowse:right()
        case nKey == K_PGDN
            oBrowse:refreshCurrent() // remove highlight from current row
            oBrowse:pageDown()
        case nKey == K_PGUP
            oBrowse:refreshCurrent() // remove highlight from current row

```

```
oBrowse:pageUp()
case nKey == K_CTRL_PGDN
  oBrowse:refreshCurrent() // remove highlight from current row
  oBrowse:goBottom()
case nKey == K_CTRL_PGUP
  oBrowse:refreshCurrent() // remove highlight from current row
  oBrowse:goTop()
case nKey == K_HOME
  oBrowse:home()
  forceleft(oBrowse)
case nKey == K_END
  oBrowse:end()
  forcerright(oBrowse)
case nKey == K_CTRL_HOME
  oBrowse:panHome()
case nKey == K_CTRL_END
  oBrowse:panEnd()
endcase
enddo
setcursor(nOldcursor) // restore previous cursor
restscreen(0, 0, maxrow(), maxcol(), cOldscreen)
use
return nil
```

```
/*
  Function: ForceLeft()
  Purpose: Force the TBrowse highlight left if we are on a separator
*/
static function forceleft(oBrowse)
if oBrowse:colPos % 2 == 0
  oBrowse:refreshCurrent() // remove highlight from current row
  oBrowse:left()
endif
return nil
```

```
/*
  Function: ForceRight()
  Purpose: Force the TBrowse highlight right if we are on a separator
*/
static function forcerright(oBrowse)
if oBrowse:colPos % 2 == 0
  oBrowse:refreshCurrent() // remove highlight from current row
  oBrowse:right()
endif
return nil
```

وظيفة "عرض العمود" colWidth()

تعيد هذه الوظيفة قيمة مسافة عرض عمود معين. وتستخدم بشكل خاص عند الحاجة لتحديد العرض دون تعيين قيمة له (أي "العرض" c:width = شيئاً ما).

والقاعدة اللغوية هي b:colWidth(<n>) حيث أن <n> تمثل العمود. وإذا كانت <n> خارج حدود البيانات أو غير متوفر فتكون القيمة المعادة الصفر.

تبين الشيفرة التالية هذه الوظيفة بإنشاء عمود لكل حقل في قاعدة البيانات ، وتوسيط اسم الحقل فوق العمود. لاحظ أنه ينبغي أولاً إضافة هدف العمود column object إلى هدف الاستعراض TBrowse قبل أن نتمكن من استخدام وظيفة "عرض العمود" b:colWidth للوصول إليه.

```
#include "inkey.ch"
function tbrow25(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, 5, maxcol())
local oColumn
local nKey
local nFields
local cField
scroll()
use (cDbfname) new
nFields := fcount()
for x := 1 to nFields
  cField := field(x)
  oColumn := TBColumnNew(, fieldblock(cField))
  oBrowse:AddColumn(oColumn)
  oColumn:heading := padc(cField, oBrowse:colWidth(x))
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  keytest(nKey, oBrowse)
enddo
use
return nil
```


وظيفة "التهيئة" () configure

تجعل هذه الوظيفة هدف الاستعراض TBrowse يعيد تهيئة نفسه ويعيد ضبط جميع المتغيرات الفورية المرتبطة بأهداف الأعمدة الموجودة ضمنه. أن هذا الإجراء ضروري لأن الاستعراض TBrowse لا يتابع كل متغير تقوم به أحد الأعمدة. فمثلاً ، إذا غيرت ضبط الألوان (كتلة الألوان c:colorBlock) فيجب عليك إعادة ضبط هدف الاستعراض وفق هذه التغييرات.

لاحظ أنه يتم استدعاء وظيفة "التشكيل": () b:configure تلقائياً كلما استدعيت وظيفة "ضبط العمود" () setColumn ، أو وظيفة "إضافة عمود" () addColumn تلقائياً أو "إدراج عمود" () insColumn ، أو "حذف عمود" () delColumn ، أو غيرت نافذة الاستعراض TBrowse (المتغيرات الفورية لإحداثيات النافذة: "رقم الصف الأعلى" b:nTop و "رقم العمود الأيسر" b:nLeft و "رقم السطر الأسفل" b:nBottom و "رقم العمود الأيمن" b:nRight). ولأن تنفيذ هذه الوظيفة يستلزم الكثير من الإجراءات ، فيفضل استدعاؤها فقط عند تغيير متغير فوري ضمن أحد أهداف العمود.

للاطلاع على مثال ذلك ، راجع بحث "ضبط العمود" b:setColumn.

وظيفة deHilite() و Hilite()

تستخدم هاتانوظيفتان عند ضبط قيمة المتغير الفوري "التظليل الآلي" إلى "غير حقيقي" (F.). (يحدد "التظليل الآلي" b:autoLite إذا كان ينبغي على الاستعراض TBrowse تظليل البيانات التي في الموضع الحالي للمؤشر تلقائياً أم لا). ويؤدي استخدام وظيفة "التظليل" () b:hilite ووظيفة "إيقاف التظليل" () b:deHilite إلى تظليل أو عدم تظليل الخلية الموجودة في الموضع الحالي للمؤشر ، على التوالي.

لن تحتاج مستقبلاً إلى إجراء التظليل بنفسك ، ومع ذلك ، إذا أردت تعديل موضع الصف يدوياً (بتغيير المتغير الفوري "موضع المؤشر في الصف" b:rowPos) فيجب عليك التظليل يدوياً تجنباً للتظليل المزدوج ، وقد ورد معنا هذا في مثال بحث "موضع المؤشر في الصف" b:rowPos .

وظيفة "حذف عمود" delColumn()

تتمكن هذه الوظيفة من حذف أعمدة من جدول الاستعراض TBrowse .

b:delColumn (<nTarget>)

حيث أن <nTarget> تعبير رقمي يمثل رقم العمود المراد حذفه.

نستخدم في الشيفرة التالية وظيفة "حذف عمود" delColumn() و "إدراج عمود" insColumn() لنقل العمود الحالي إلى العمود رقم (١):

```
local c := b:getColumn(b:colPos)
b:delColumn(b:colPs)
b:insColumn(1, c)
```

للاطلاع على مثال آخر على "حذف عمود" delColumn() راجع فقرة "ضبط الأعمدة" setColumn() .

التثبيت الجبري forceStable()

لقد أضيفت هذه الوظيفة في كليب الإصدار 5.2 للقيام بالتثبيت الكامل لهدف الاستعراض Tbrowse . وهو مساوٍ حلقة التوازن التقليدية لكنه ينفذ عمله بسرعة أكبر. فإذا كنت تستخدم كليب 5.2 ، فمن الأفضل استخدام وظيفة "التثبيت الجبري" forceStable() بدلاً من حلقة DO WHILE في وظيفة "التأسيس" b:stsbilize().

استرجاع هدف عمود getColumn()

تسترجع هذه الوظيفة عمود الاستعراض TBrowse بموجب قيمة رقمية لمتغير مستقل ، تمثل هذه القيمة ترتيب العمود في هدف الاستعراض TBrowse . وسنستخدم هذه الوظيفة مراراً للحصول على رقم للعمود الذي نكون فيه.

إدراج عمود insColumn()

تستخدم هذه الوظيفة لإدراج أعمدة جديدة. والقاعدة اللغوية لذلك هي:

b:insColumn(<nTarget> , <oSource>)

حيث أن <nTarget> هو تعبير رقمي يمثل الموضع الذي سندرج فيه العمود الجديد.

والمتغير <oSource> الهدف TBColumn المراد إدراجه في ذلك الموضع.

نقوم في المثال التالي بإدراج عمود جديد كعمود رقم (١).

b:insColumn(1, TBColumnNew("New Column", { | | "New Column" }))

للاطلاع على مثال آخر عن "إدراج عمود" () insColumn راجع بحث "ضبط الأعمدة" () setColumn.

الوظيفة Invalidate()

تسبب هذه الوظيفة قيام تثبيت TBrowse التالية بإعادة رسم كافة بيانات TBrowse (بما في ذلك الترويسات ، والتذييلات ، وكافة صفوف البيانات). وليس لهذه الوظيفة أي تأثير على القيم في صفوف البيانات ، بل إنها تسبب تحديث العرض أثناء عملية التأسيس التالية فحسب. وهي لذلك أسرع بقليل من وظيفة "تجديد كافة البيانات" () b:refreshAll.

ومع ذلك يجب استخدام وظيفة "تجديد كافة البيانات" (b:refreshAll) بدلاً من هذه الوظيفة إذا كان من الضروري استحضار البيانات من مصدر البيانات الأساسية.

وظيفة refreshAll() و refreshCurrent()

لقد ذكرنا آنفاً أن الاستعراض TBrowse ومصدر البيانات الخاص بك كيانان منفصلان تماماً. حيث يقوم هدف الاستعراض TBrowse فقط بدور الاتصال البيئي مع البيانات فإذا غيرت البيانات ، فلن يعرف هدف الاستعراض TBrowse ذلك بأي حال من الأحوال. ولذلك تستخدم هاتين الوظيفتين لإخبار هدف الاستعراض TBrowse بأن عليه إعادة عرض كافة البيانات أو جزء منها.

تؤشر الوظيفة "تجديد كافة البيانات" (refreshAll) كافة صفوف البيانات بأنها غير صحيحة ، مما يؤدي إلى إعادة رسم الشاشة بأكملها عندما تعود إلى حلقة التأسيس. أما وظيفة "تجديد الصف الحالي" (refreshCurrent) فتؤشر فقط على البيانات التي في الصف الحالي الموجود فيها المؤشر بأنها غير صحيحة ، مما يؤدي إلى إعادة عرض الصف في المرة التالية التي تستدعي فيها وظيفة التأسيس (b:stabilize).

من الضروري استخدام هاتين الوظيفتين عند تغيير البيانات التي في هدف الاستعراض TBrowse. وأفضل مثال على استخدامهما هو إجراءات الإضافة والتعديل/التحرير. فإذ إضافة سجل يستدعي وظيفة "تجديد كافة البيانات" (b:refreshAll) لأنه ليس هناك طريقة لمعرفة مكان ظهور السجل الجديد ، خاصة إذا كان لدينا ملف فهرس واحد أو أكثر. إن تعديل/تحرير سجل واحد قد يتطلب فقط استخدام وظيفة "تجديد الصف الحالي" (refreshCurrent) لأنه لن يتأثر إلا هذا الصف (ما لم نعدل مفتاح الحقل في فهرس التحكم).

يبين المثال التالي هاتين الوظيفتين. اضغط مفتاح **Enter** لتعديل خلية. إذا عدلت حقل مفتاح ضمن الفهرس فسيستدعي وظيفة "تجديد كافة البيانات"

() refreshAll لأن السجل الحالي قد يحتاج إلى إعادة ترتيب على الشاشة. وإذا عدلت حقلاً ليس جزءاً من دليل الفهرس فسيكفي استدعاء وظيفة "تجديد الصف الحالي" () refreshCurrent. لاحظ أن هذه القاعدة المنطقية تعمل لأن كل اسم حقل مخزون في المتغير الفوري "الرويسة" c:heading. فإذا أردت تغيير ترويسات الأعمدة ، فيجب عليك تخزين أسماء الحقول في حيز "الشحنة" cargo.

يمكنك أيضاً تحرير فهرسين ، وبهذه الحالة ستستخدم مفتاحي [Alt]-[I] للتقلب بينهما. عند الانتقال لفهارس التحكم ، تستخدم وظيفة "تجديد كافة البيانات" () refreshAll لتقوم بإعادة عرض الشاشة بأكملها لتظهر الترتيب الجديد.

ملاحظة خاصة بالشبكات

إن جميع أمثلتنا الخاصة بـ: GET ضمن جدول العرض TBrowse أو معظمها يعمل مباشرة في حقل قاعدة البيانات. وفي مثل هذه الحالات يجب علينا التأكد من إقفال السجل قبل استدعاء وظيفة "القراءة المشروطة" () ReadModal. ومع ذلك يختلف هذا المثال قليلاً حيث أن محتويات الحقل تنسخ سلفاً إلى متغير ، وهذا يتيح لنا إقفال السجل قبل تحديث حقل قاعدة البيانات مباشرة ، وليس قبل عملية القراءة READ. وتحدث هذه الطريقة مشاكل عدة مثل "الإقفال لمدة مؤقتة" "Lunch-time lock" (عندما يتعد المستخدم عن طرفيته أثناء القراءة READ).

فكرة مفيدة

إذا كنت تعمل ضمن بيئة شبكة متعددة المستخدمين ، فيمكنك استخدام وظيفة "تجديد كافة البيانات" () refreshAll ، لتجديد عرض البيانات على الشاشة باستمرار. تجدد بيانات الشاشة ، في المثال التالي كل عشر ثوان بانتظار ضغط أي مفتاح. ويمكنك أيضاً استخدام مؤشرات لتجديد الشاشة فقط عند يتم تعديل قاعدة البيانات بواسطة جهاز (محطة عمل طرفية) آخر. وحتى هذا الاستخدام البسيط يعتبر أفضل من استخدام DBEDIT()


```
#include "inkey.ch"
#include "setcurs.ch"

// this manifest constant sets # of seconds for auto-refresh
#define REFRESH_TIME 10
function tbrow26(cDbfname, cNtx1, cNtx2)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local nKey
local nSeconds
local nFields
local nOldRow
local nOldCol
local cField
set scoreboard off // yuch!
setcursor(SC_NONE)
if cDbfname == NIL .or. cNtx1 == NIL
    ? "Syntax: BROWSER <dbf> <n timer> [<n timer>]"
    return nil
endif
scroll()
if cNtx2 <> NIL
    use (cDbfname) index (cNtx1), (cNtx2) new
else
    use (cDbfname) index (cNtx1) new
endif
nFields := fcount()
for x := 1 to nFields
    cField := field(x)
    oBrowse:AddColumn( TBColumnNew(cField, fieldblock(cField)) )
next
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    if nKey == 0
        nOldRow := row()
        nOldCol := col()
        @ oBrowse:nTop - 1, oBrowse:nRight - 11 say ;
            if(deleted(), "<deleted>", replicate(chr(196),9))
        setpos(nOldRow, nOldCol)
        nSeconds := seconds()
        // use another loop so we can refresh while waiting for keypress
        do while ( nKey := inkey() ) == 0
            if seconds() - nSeconds > REFRESH_TIME
                nSeconds := seconds()
                oBrowse:refreshAll()
                dispbegin()
            endif
        enddo
    else
        nKey := inkey()
    endif
enddo
```

```

        do while ! oBrowse:stabilize()
        enddo
        dispend()
    endif
enddo
endif
if ( nKey == K_ENTER .or. (nKey > 32 .and. nKey <= 255) ) ;
    .and. oBrowse:stable
    if nKey <> K_ENTER
        keyboard chr(nKey)
    endif
    if editcell(oBrowse)
        if oBrowse:getcolumn(oBrowse:colpos):heading $ upper(indexkey(0))
            oBrowse:refreshAll()
        else
            oBrowse:refreshCurrent()
        endif
    endif
    // swap between controlling indexes if we specified two of them
elseif cNtx2 <> NIL .and. nKey == K_ALT_I
    set order to if(indexord() == 1, 2, 1)
    oBrowse:refreshAll()
elseif nKey == K_DEL
    if deleted()
        recall
    else
        delete
    endif
else
    keytest(nKey, oBrowse)
endif
enddo
use
return nil
/*

```

Note: many examples of GETs within a TBrowse window operate directly on the database field. Here we copy the field contents to a variable, which would allow us to discard the changes if necessary.

```

*/
static function editcell(oBrowse)
local nKey
local lReadexit := readexit(.t.)
local oColumn := oBrowse:getColumn(oBrowse:colPos)
local oldvalue := eval(oColumn:block)
local v := oldvalue
local oldcursor := setcursor(SC_NORMAL)

```

```
local IRetval := .f.  
readmodal( { getnew(Row(), Col(), ;  
              { | _1 | if(pcount() == 0, v, v := _1) }, ;  
              oColumn:heading, '@K', oBrowse:colorSpec) } )  
setcursor(oldcursor)  
if v <> oldvalue          // i.e., variable was changed  
  if rlock()              // make sure record is locked  
    eval(oColumn:block, v) // this changes the field  
    unlock  
    IRetval := .t.  
  endif  
endif  
readexit(IRetval)  
nKey := lastkey()  
if nKey <> K_ENTER  
  keyboard chr(nKey)  
endif  
return IRetval
```

وظيفة "ضبط الأعمدة" setColumn

تقوم هذه الوظيفة باستبدال أهداف "عمود الاستعراض" TBColumn بأخرى. وتقبل قيمتين لتغيرين مستقلين:

setColumn(<nColumn> , <oColumn>)

وحيث أن المتغير <nColumn> قيمة رقمية تمثل العمود المراد استبداله.

وحيث أن <oColumn> هدف عمود الاستعراض TBColumn الذي سيحل محل <nColumn>.

نستخدم في المثال التالي العديد من هذه الوظائف لتعليم (تظليل) ونسخ ونقل الأعمدة. فنستخدم وظيفة "ضبط الأعمدة" setColumn() ، ووظيفة "استرجاع هدف عمود" getColumn() ، ووظيفة "تجديد كافة البيانات" refreshAll() ، ووظيفة "التشكيل" configure() ، إضافة إلى المتغيرين الفوريين "موضع المؤشر في العمود" b:colPos و "تحديد الألوان" c:defColor. ويمكننا أيضاً إدراج وحذف الأعمدة باستخدام وظيفتي "إدراج عمود" insColumn() و "حذف عمود" delColumn() .

لتعليم عمود ونسخه أو نقله ، انتقل إلى ذلك العمود واضغط مفتاح **[Enter]** ، فيعرض بألوان معكوسة. حرك المؤشر إلى العمود المراد استبداله واضغط المفتاح **[C]** (نسخ) أو **[M]** نقل. ولإدراج عمود في موضع معين ، اضغط مفتاح **[Ins]** (إدراج) ، فتعرض أمامك قائمة من أسماء الحقول ، ويعرض الاسم الذي تختاره في العمود الجديد. وحذف العمود الحالي ، اضغط مفتاح **[Del]** (حذف).

ولتقليص أو توسيع العمود الحالي اضغط مفتاحي **[Ctrl]-[←]** و **[Ctrl]-[→]** على التوالي.

```
#include "box.ch"
#include "inkey.ch"
function tbrow27(cDbfname)
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey
local lTagged := .f.
local aFields
local nFields
local nSourceCol
setcursor(0)
scroll()
use (cDbfname) new
nFields := fcount()
aFields := array(nFields) // initialize fieldnames array
for x := 1 to nFields
    aFields[x] := field(x)
    oBrowse:addColumn( TBColumnNew(aFields[x], fieldblock(aFields[x])) )
    // sneaky trick to actually stick the value of the column width
    // into the column:width instance variable (so we can shrink/expand)
    oBrowse:getColumn(x):width := oBrowse:colWidth(x)
next
do while nKey <> K_ESC
    dispbegin()
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
    dispend()
    if nKey == 0
        nKey := inkey(0)
    endif
    oColumn := oBrowse:getColumn(oBrowse:colPos)
```

```

do case
  case nKey == K_ENTER .and. ! ITagged
    // highlight this entire column by changing c:defColor
    oColumn:defColor := { 2, 2 }

    // save current column position for future reference
    nSourceCol := oBrowse:colPos

    // must reconfigure TBrowse object to reflect color change
    oBrowse:configure()
    ITagged := .t.

  case upper(chr(nKey)) == "C" .and. ITagged
    // de-highlight source column by changing c:defColor
    oColumn := oBrowse:getColumn(nSourceCol)
    oColumn:defColor := { 1, 2 }
    oBrowse:setColumn(oBrowse:colPos, oColumn)
    ITagged := .f.

  case upper(chr(nKey)) == "M" .and. ITagged
    oColumn := oBrowse:getColumn(nSourceCol)
    oColumn:defColor := { 1, 2 }
    oBrowse:setColumn(oBrowse:colPos, oColumn)
    // delete source column
    oBrowse:delColumn(nSourceCol)
    ITagged := .f.

  case nKey == K_INS
    if ! empty(x := pickfield(aFields))
      oBrowse:insColumn(oBrowse:colPos, TBColumnNew(x, fieldblock(x)))
      oBrowse:getColumn(oBrowse:colPos):width := ;
      oBrowse:colWidth(oBrowse:colPos)
    endif
  case nKey == K_DEL
    oBrowse:delColumn(oBrowse:colPos)
  case nKey == K_CTRL_LEFT .and. oColumn:width > 1
    oColumn:width--
    oBrowse:configure()
  case nKey == K_CTRL_RIGHT
    oColumn:width++
    oBrowse:configure()
  otherwise
    keytest(nKey, oBrowse)
endcase
enddo
use
return nil

```



```

static function pickfield(aFields)
local midcol := maxcol() / 2
local cBuffer := savescreen(0, midcol - 6, maxrow() - 1, midcol + 6)
local nSel
@ 0, midcol - 6, maxrow() - 1, midcol + 6 box B_SINGLE + ''
nSel := achoice(1, midcol - 5, maxrow(), midcol + 5, aFields)
restscreen(0, midcol - 6, maxrow() - 1, midcol + 6, cBuffer)
return if(nSel > 0, aFields[nSel], NIL)

```

عند تعليم عمود ما يعدل المتغير الفوري "تحديد الألوان" c:defColor بحيث يعرض
بالوان معكوسة ، وبذلك يكون من الضروري إعادة تشكيل هدف الاستعراض
TBrowse بواجب التشكيل () b:configure بحيث يظهر اللون الجديد للعمود على
الشاشة.

إن أول ما يجب عمله بعد اختيار العمود المراد استبداله هو إعادة ضبط لون
العمود المصدر (أيضاً باستخدام المتغير الفوري defColor) ، وإلا سيعرض على
الشاشة عمودان مقلوبان مما يسبب التشويش. ثم نمرر رقم العمود المراد استبداله ،
وهدف عمود المصدر المحفوظ سابقاً ، إلى وظيفة "ضبط الأعمدة" () b:setColumn.

نسخ العمود () CloneColumn

عند استخدام وظيفة "ضبط الأعمدة" () b:setColumn على النحو المبين في المثال
الآخر لا يتم نسخ العمود الأصلي ، بل يتم إنشاء مرجع إضافي له. وأصعب ما في ذلك
أنك إن غيرت متغيراً فورياً مرتبطاً بأحد العمودين (الأصلي أو المرجع الإضافي)
فستضطر إلى تغييرهما.

إن عملية إنشاء المرجع الإضافي معقدة وقد تؤدي إلى إعادة ترتيب العمود.
ولإدراك ذلك ، نعد المثال الأخير لنسخ عمود ما إلى موضع آخر ، ثم استخدم المفاتيح
[Ctrl] + [→] و [Ctrl] + [←] ، لتعديل عرض العمود الأصلي ، فتلاحظ أن العرض قد
تغير في كلا العمودين الأصلي والنسخة.

لذلك من الضروري إيجاد طريقة "نسخ" عمود. بحيث يكون العمود الجديد منفصلاً تماماً عن الأصلي. ومع أن المثال التالي ليس تاماً إلا أنه يبين طريقة القيام بذلك باستخدام وظيفة "نسخ العمود" CloneColumn().

تنشئ وظيفة "نسخ العمود" CloneColumn() عموداً جديداً باستخدام كتلة ترويسة العمود المصدر ، ثم ينسخ كافة المتغيرات الفورية الأخرى من العمود المصدر إلى العمود الذي تم إنشاؤه. وأخيراً يستدعي وظيفة "ضبط الأعمدة" b:setColumn() لتعين هذا العمود الجديد في الموضع الذي حددته. يتضمن الإصدار 5.2 من كليبر ، فحص الصفء NIL وهو ضروري بسبب ميزة تدقيق الأخطاء في كليبر 5.2 ، التي تعين نوع بيانات غير صحيح invalid إلى متغير فوري للاستعراض TBrowse أو لعمود الاستعراض TBColumn. (مع أن معظم المتغيرات الفورية لعمود الاستعراض TBColumn تعطي قيمة الصفء NIL ، إلا أن كليبر لا يسمح لك باستهلاكهم بالقيمة NIL).

```
static function clonecolumn(oBrowse, oSource)
local nTarget := oBrowse:colPos
local oTarget := TBColumnNew(oSource:heading, oSource:block)
if oSource:footing <> NIL
  oTarget:footing := oSource:footing
endif
oTarget:cargo := oSource:cargo
if oSource:colSep <> NIL
  oTarget:colSep := oSource:colSep
endif
if oSource:colorBlock <> NIL
  oTarget:colorBlock := oSource:colorBlock
endif
if oSource:footSep <> NIL
  oTarget:footSep := oSource:footSep
endif
if oSource:headSep <> NIL
  oTarget:headSep := oSource:headSep
endif
oTarget:defColor := oSource:defColor
oTarget:width := oSource:width
oBrowse:setColumn(nTarget, oTarget)
return nil
```

ملاحظة

يمكن معالجة الأهداف كمصفوفة. وسنستخدم هذا المبدأ في النسخة التالية المعدلة من وظيفة "نسخ العمود" CloneColumn(). وهذا أشمل لأنه لا حاجة لتشفير أسماء المتغيرات الفورية في البرنامج. كما أنه يتجاوز ميزة فحص نوع البيانات المضافة في الإصدار 5.2 من كليبر.

```
static function colnecolumn(oBrowse, oSource)
local oTarget := TBColumnNew( )
local y := len(oSource)
local x
for x := 1 to y
    oTarget[x] := oSource[x]
next
oBrowse:setColumn(oBrowse:colPos, oTarget)
return nil
```

وظيفة "التثبيت" stabilize()

تحدد هذه الوظيفة كيف يعرض جدول الاستعراض TBrowse البيانات على الشاشة فيعيد قيمة "حقيقي" (.T.) إذا كان هدف جدول الاستعراض ثابتاً وكانت جميع البيانات معروضة ضمن حدود الاستعراض بشكل صحيح ، أو قيمة "غير حقيقي" (.F.) إذا انحرف أي شيء منها. وقد مرّ معنا سابقاً العديد من الأمثلة على استخدام هذه الوظيفة.

أمثلة متقدمة على ميزات الاستعراض TBrowse

مر معنا كيفية معالجة كتل الانتقال في الاستعراض TBrowse لتحديد عرض مجموعة جزئية من البيانات. ومع ذلك قد يتطلب الأمر أن يقوم المستخدم بتحديد المجموعات الجزئية الخاصة به ، ويمكن إنجاز ذلك بسهولة حيث يمكننا تجميع كتل الشيفرة أثناء عملية التشغيل. وفيما يلي خطوات هذا الإجراء:

١- نحدد في أعلى البرنامج "القيمة العليا" HIVAL و "القيمة الدنيا" LOWVAL كمتغيرين ثابتين على عرض الملف ، ويمثل هذان المتغيران القيمة العليا والقيمة الدنيا للمجموعة الجزئية من البيانات.

٢- ثم نعد مفتاح `[Enter]` لاستدعاء وظيفة "مرشح زائف" `PseudoFilt()` والتأكد من تمرير هدف الاستعراض TBrowse كمتغير.

٣- تمكّنك وظيفة "مرشح زائف" `PseudoFilt()` من إدخال القيمة العليا والقيمة الدنيا للنطاق. وتفرض أنك لا تخرج باستخدام مفتاح `[Esc]` ، سيقوم معالج الأخطاء المبيت في تتبع أي أخطاء لغوية في كتلة الشيفرة. وقد وجدنا أن هذا ضروري وذلك لأنه من السهل جداً وجود سلسلة حرفية لا يمكن تجميعها بشكل صحيح في كتلة الشيفرة `code block`.

٤- تقوم هذه العبارات التالية بتنشيط معالج الأخطاء.

```
bNewhandler = { | oError | blockhead(oError, bOldhandler) }
bOldhandler = errorblock(bNewhandler)
```

تخدم وظيفة "كتلة الترويسة" `BlockHead()` غرضاً واحداً وهو: تدقيق وتصحيح الأخطاء اللغوية المتضمنة في السلسلة التي أدخلتها.

٥- بمجرد أن يكون معالج الأخطاء جاهزاً ، نحاول وظيفة "مرشح زائف" `PseudoFilt()` تجميع كتل الانتقال الثلاث. وتستخدم الوظيفتان "الانتقال إلى

كتلة أعلى " b:goToBlock() و "الانتقال إلى كتلة أسفل" b:goBottomBlock وظيفة "البحث في قاعدة البيانات" DBSEEK(). لاحظ أن وظيفة "البحث في قاعدة البيانات" DBSEEK() تقبل متغيراً منطقياً ثانياً ويتحكم هذا المتغير الثاني ببرنامج (SOFTSEEK). إذا مرت قيمة "حقيقي" (T.) سينفذ برنامج كليبر البحث. وبهذا لن نضطر للقيام بضبط "بحث البرنامج" على وضع تشغيل SOFTSEEK ON ثم إعادته إلى حالته الأولى.

٦- لاحظ الصيغة المستخدمة في وظيفة "الانتقال إلى كتلة أعلى" b:goToBlock(). نستخدم فيها قيمة النهاية كما هي ، باستثناء آخر رمز في الجهة اليمنى الذي يزيد بقيمة شيفرة آسكي ASCII واحدة.

٧- والسبب وراء ذلك هو أنه لو كان لدينا ، فرضاً ، عشر سجلات محتوية لقيمة النهاية ، فإننا نريد أن تكون جميعها مرئية. لذلك ينبغي وضع مؤشر السجل وراء آخر سجل مباشرة ، ثم نقفز إلى الخلف (باستخدام وظيفة "تجاوز قاعدة البيانات" DBSKIP()).

٧- تشكل وظيفة "تجاوز كتلة" b:skipBlock() لاستدعاء وظيفة التجاوز العادية. ويمرر لهذه الوظيفة متغيرين: قيمة الانتقال الداخلي (ورد آنفاً) ، وكتلة الشيفرة التي تعيد قيمة دليل الفهرس النشط. ثم تقيم الوظيفة بعد ذلك كتلة الشيفرة وتقارنها بالقيمتين العليا والدنيا للتأكد من أننا مازلنا ضمن النطاق المحدد للبيانات.

٨- بفترض أن كتلة البيانات تجمع بشكل صحيح ، فينبغي بعد ذلك ضبط "المرشح filter" باستدعاء وظيفة "الانتقال إلى أعلى" b:goTop() (المساوية لأمر "الانتقال إلى أعلى" GO TOP).

بهذه الإجراءات يمكننا عرض المجموعة الجزئية التي اخترناها. وإذا أردت عرض سجلات مختلفة ، اضغط مفتاح **[Enter]**. ستظهر أمامك القيمتان الحاليتان العليا والدنيا اللتان يمكنك تغييرهما أو مسحهما. (وفي حالة مسحهما ، ستعرض أمامك قاعدة البيانات بأكملها ثانيةً).

نستخدم في هذا المثال أيضاً وظيفة () NTXPOS لعرض مسطرة حالة عمودية على الجانب الأيمن من نافذة جدول الاستعراض TBrowse. ولعرض الموضع النسبي الملائم ، حتى ضمن المجموعة الجزئية من البيانات ، فإننا نستخدم "رقم بداية السجل" nStartRecNo و "رقم نهاية السجل" nEndRecNo. ويخزن هذان المتغيران الثابتان على عرض الملف ، المعادلة (فرق القيمة) لرقم السجل الأول ورقم السجل الأخير على التوالي في المجموعة الجزئية من البيانات.

```
// filename:TBROW30.PRG
external dbskip, dbseek

#include "inkey.ch"
#include "box.ch"
#include "error.ch"

//----- display message on row #1 regarding subset
#define DisplayMessage( <msg> ) => ;
    setpos(1, 1) ; ;
    dispout(padc( <msg>, maxcol() - 2), MAINCOLOR)

#define RESETMSG "Viewing all records -- press Enter for subset"
#define MAINCOLOR "+W/B"

static cStartval // storage space for goTopBlock
static cEndval // storage space for goBottomBlock
static nStartRecNo // offset for first record # in subset
static nEndRecNo // last record # in subset

function tbrow30(cDbfname, cNtxname)
local oBrowse
local oColumn
local cField
local x
```

```

local y
local cOldcolor := setcolor()
local nKey
local nOldcursor := setcursor(0)
local lOldscore := set(_SET_SCOREBOARD, .f.)
local cOldscrm := savescreen(0, 0, maxrow(), maxcol())
local blIndex
local nStatusRow
local aMorekeys := { { K_ENTER, { | oBrowse | pseudofilt(oBrowse) } } }
if cDbfname == NIL
    return .f.
else
    use (cDbfname) new
    //----- activate index if one was specified
    if cNtxname <> NIL
        dbsetindex(cNtxname)
        blIndex := &("{ || " + indexkey(0) + "}")
    else
        //----- for the purpose of this demo, locate first character field
        //----- and create our primary index based on that field
        ? "creating temporary index for this demo..."
        y := fcount()
        x := 0
        do while ++x <= y .and. type(cField := field(x)) <> "C"
        enddo
        blIndex := &("{ || " + cField + "}")
        dbCreateIndex('blahblah', cField, blIndex)
        //----- we must close then re-open this temporary index so
        //----- that NTXPOS() can have proper access to it
        dbClearIndex()
        dbSetIndex('blahblah')
    endif
endif
scroll()
@ 0, 0, maxrow() - 1, maxcol() box B_DOUBLE + ' ' color MAINCOLOR
oBrowse := TBrowseDB(2, 1, maxrow() - 2, maxcol() - 1)
oBrowse:colorSpec := MAINCOLOR + ',GR+/N,N/N,N/N,N/W'
oBrowse:headSep := chr(205) + chr(209) + chr(205)
oBrowse:colSep := chr(32) + chr(179) + chr(32)
oBrowse:footSep := chr(205) + chr(207) + chr(205)
oBrowse:cargo := blIndex
for x := 1 to 3
    cField := field(x)
    oColumn := TBColumnNew(cField, fieldblock(cField))

```

```

    oColumn:footing := oColumn:heading
    oBrowse:AddColumn(oColumn)
next
DisplayMessage(RESETMSG)

//----- draw elevator bar on right side of box for vertical scrollbar
//----- and initialize the associated variables
@ oBrowse:nTop, oBrowse:nRight + 1, ;
  oBrowse:nBottom, oBrowse:nRight + 1 box replicate(chr(176), 9) ;
  color MAINCOLOR
nStatusRow := oBrowse:nTop + 1
nStartRecNo := 0
nEndRecNo := lastrec()

do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
    if nKey == 0
      ShowBar(oBrowse, @nStatusRow)
      nKey := inkey(0)
    endif
    keytest(nKey, oBrowse, aMorekeys)
  enddo
setcursor(nOldcursor)          // restore previous cursor
set(_SET_SCOREBOARD, lOldscore) // restore previous SCOREBOARD
setcolor(cOldcolor)            // restore previous color
restscreen(0, 0, maxrow(), maxcol(), cOldscm)
use
ferase('blahblah.ntx')
return nil

static function pseudofilt(oBrowse)
static bOldTopBlock
static bOldBottBlock
static bOldSkipBlock
local cOldcolor := setcolor("+W/RB")
local bOldhandler
local bNewhandler
local cOldscm := savescreen(7, 19, 10, 60)
local getlist := {}
@ 7, 19, 10, 60 box B_SINGLE + ''
cStartval := if(cStartval == NIL, padr(eval(oBrowse:cargo), 20), padr(cStartval, 20))
cEndval := if(cEndval == NIL, space(20), padr(cEndval, 20))
@ 8, 21 say "Enter start value:" get cStartval picture '@K'
@ 9, 21 say "Enter end value: " get cEndval picture '@K' ;
      when (cEndval := cStartval) == cStartval

```

```

setcursor(1)
read
setcursor(0)
restscreen(7, 19, 10, 60, cOldscm)
cStartval := trim(cStartval)
cEndval := trim(cEndval)
setcolor(cOldcolor)
if lastkey() <> K_ESC .and. ! empty(cStartval) .and. ! empty(cEndval)
  bNewhandler := { | e | blockhead(e, bOldhandler) }
  bOldhandler := errorblock(bNewhandler)
  bOldTopBlock := oBrowse:goTopBlock
  bOldBottBlock := oBrowse:goBottomBlock
  bOldSkipBlock := oBrowse:skipBlock
  begin sequence
    oBrowse:goTopBlock := &("{ || dbseek("'" + cStartval + "'", .t.)}")
    oBrowse:goBottomBlock := &("{ || dbseek("'" +
      left(cEndval, len(cEndval) - 1) + ;
      chr(asc(right(cEndval, 1)) + 1) + ;
      "'", .t.), dbskip(-1) }")
    oBrowse:skipBlock := { | nSkipCnt | Gilligan(nSkipCnt, ;
      &("{ || " + indexkey(0) + "}") ) }
    eval(oBrowse:goBottomBlock)
    nEndRecno := ntxpos(indexord(), recno())
    eval(oBrowse:goTopBlock)
    nStartRecno := ntxpos(indexord(), recno())
    nEndRecno := nStartRecno

    //----- force "filter" to be set by "going top"
    oBrowse:goTop()
    DisplayMessage("Now viewing data between " + cStartval + ;
      " and " + cEndval)
  end
  errorblock(bOldhandler) // reset previous error handler
else
  //----- reset prior movement blocks
  if bOldTopBlock <> NIL
    oBrowse:goTopBlock := bOldTopBlock
    oBrowse:goBottomBlock := bOldBottBlock
    oBrowse:skipBlock := bOldSkipBlock
    nStartRecno := 0
    nEndRecno := lastrec()
    //----- force "filter" to be set by "going top"
    oBrowse:goTop()
  endif
  cStartval := cEndval := " // reset subset range holders

```



```
    DisplayMessage(RESETMSG)
endif
return nil
```

```
static function blockhead(e, bOldhandler)
if e:gencode() == EG_SYNTAX
    alert("Error in code block syntax")
    break
endif
return eval(bOldhandler, e)
```

```
static function gilligan(nSkipCnt, bVal)
local nMovement := 0
do case
    //----- no movement
    case nSkipCnt == 0
        skip 0
    //----- moving forward
    case nSkipCnt > 0
        do while nMovement < nSkipCnt .and. eval(bVal) <= cEndval .and. ! eof()
            skip 1
            nMovement++
        enddo
        //----- make sure that we are within range - if not, move backward
        do while (eval(bVal) > cEndval .or. eof()) .and. ! bof()
            skip -1
            nMovement--
        enddo
        if bof()      // no data in range... fall out
            // keyboard chr(K_ESC)
        endif
    //----- moving backward
    case nSkipCnt < 0
        do while nMovement > nSkipCnt .and. eval(bVal) >= cStartval
            skip -1
            if bof()
                exit
            endif
            nMovement--
        enddo
        //----- make sure that we are within range -- if not, move forward
        do while eval(bVal) < cStartval .and. ! eof()
            skip
```



```

        nMovement++
    enddo
    if eof()      // no data within range... fall out
        // keyboard chr(K_ESC)
    endif
endcase
return nMovement

static function ShowBar(oBrowse, nStatusRow)
local nEle
local nOldRow := row()
local nOldCol := col()

//----- determine relative position
nEle := ntxpos(indexord(), recno()) - nStartRecNo

//----- determine if status row has changed
if nStatusRow <> oBrowse:nTop + int((nEle / nEndRecNo) * ;
    (oBrowse:nBottom - oBrowse:nTop))
    dispbegin()
    //----- first, blank out previous status bar
    @ nStatusRow, oBrowse:nRight + 1 say chr(176) color MAINCOLOR
    //----- then recalculate position of status bar
    nStatusRow := oBrowse:nTop + int((nEle / nEndRecNo) * ;
        (oBrowse:nBottom - oBrowse:nTop))
    //----- finally, redraw it
    @ nStatusRow, oBrowse:nRight + 1 say chr(219) color MAINCOLOR

    dispend()

endif
setpos(nOldRow, nOldCol)
return nil

```

إذا كان لديك معايير استعمال query criteria لاتتوافق مع أحد الفهارس ، فيجب عليك الرجوع إلى الأمر القديم "ضبط المرشح" SET FILTER. وهذا سابقاً ، أما الآن فسنعتمد على المتغير الفوري "الشحنة" Cargo في جدول الاستعراض TBrowse. سيخزن المتغير الفوري b:cargo مصفوفة تحتوي أرقام السجلات المطابقة لمعيار الاستعلام الخاص بنا. ومع أنه مازال علينا أن نعالج الملف بأكمله لإيجاد كافة

السجلات المطابقة مثلما كنا نفعل مع "المرشح" ، إلا أن الميزة هنا هي أننا بعد الانتهاء من إنشاء هذه المصفوفة يمكننا القفز بسرعة من سجل إلى آخر والانتقال إلى أعلى وأسفل البيانات الصحيحة. والأهم من ذلك هو أننا نخلصنا من الوقت الضائع عند محاولة الانتقال وراء إحدى نهايتي البيانات الصحيحة.

أثناء الدوران في قاعدة البيانات بحثاً عن السجلات المطابقة ، ستظهر معلومات أمام المستخدم (رقم السجل الحالي ، عدد السجلات المطابقة حتى ذلك الوقت ، إجمالي عدد السجلات في قاعدة البيانات) ، وبهذا يعرف المستخدم أن عملية البحث جار تنفيذها.

سنشكل أيضاً المتغيرين الفوريين "الانتقال إلى كتلة أعلى" goTopBlock و "الانتقال إلى كتلة أسفل" goBottomBlock للقفز إلى السجلين الأول والآخر في مصفوفة الشحنة cargo array. وينبغي أن تغير الكتل أيضاً مؤشر عنصر المصفوفة وفقاً لذلك.

إن القاعدة المنطقية "غير القياسية" الأخرى في هذا المثال موجودة ضمن وظيفة "تجاوز كتلة" skipBlock. تقوم الوظيفة "تجاوز كتلة" skipBlock التقليدية بنقل مؤشر السجل وجعله يتجاوز سجلاً واحداً في كل مرة ، ولكن هذا لا يخدم الغرض من القاعدة المنطقية. وبما أننا حفظنا أرقام السجلات الصحيحة على التوالي في مصفوفة الشحنة ، فمن السهل الانتقال GO إلى السجل الصحيح التالي بدلاً من تجاوز السجلات عشوائياً.

لاحظ أن كلا من nCurrEle و nMaxEle هما متغيران ثابتان على مدى الملف ، ولذلك لا حاجة لتمريضهما كمتغيرين في وظيفة () Gilligan. ولكن يجب تمرير هدف جدول الاستعراض TBrowse ليتمكننا الوصول إلى مصفوفة الشحنة الخاص بها.

ومع أن الوظيفة مفيدة وقوية جداً ، إلا أنها ليست حلاً لكافة الحالات:

- إذا أردت ترشيح عدد قليل جداً من السجلات الصحيحة (مثلاً ١٠ سجلات من أصل ١٠,٠٠٠) فقم بضبط المرشح SET FILTER.
- لا يقبل أكثر من ٤٠٩٦ سجلاً مطابقاً للمعيار المحدد ، وهذا بسبب تحديد كليبر لعدد العناصر في المصفوفة. ولذلك تم تضمين قاعدة منطقية لتحديد أقصى عدد من السجلات المطابقة.

```
// filename : TBROW31.PRG
#include "inkey.ch"
#include "setcurs.ch"
#include "box.ch"

#define TOPRECORD    oBrowse:cargo[1]
#define BOTTOMRECORD ATail(oBrowse:cargo)

#define TEST // to compile test stub

#ifdef TEST
function tbrow31
local x
dbcreate('tbrow31.dbf', { { "NAME", "C", 10, 0 } } )
scroll()
use tbrow31 new
for x := 1 to 1200
    append blank
    if x % 50 == 0
        tbrow31->name := "GRUMPFISH"
    else
        tbrow31->name := "HAPPYFISH"
    endif
next
for x := 1 to 500
    append blank
    tbrow31->name := "HAPPYFISH"
next
browfilt("trim(tbrow31->name) == 'GRUMPFISH'")
use
ferase('tbrow31.dbf')
return nil

#endif // end test stub
```

```

function browfilt(cCondition)
local x
local oBrowse
local oColumn
local nKey
local nOldcursor := setcursor(0)
local cOldscrn := savescreen(0, 0, maxrow(), maxcol())
local cOldcolor := setcolor("+w/r")
local bCondition := &("{ | | " + cCondition + "}")
local nMaxEle := 0           // maximum array elements
local nCurrEle := 1         // current array element

//----- determine first record matching query criteria
go top
do while ! eval(bCondition) .and. ! eof()
    skip
enddo
//----- no data matching query criteria
if eof()
    ? "No records found matching query criteria"
    return nil
endif
//----- user feedback box so they know something's goin' on
DispBox(10, 23, 14, 56, B_SINGLE + ' ')
@ 11, 25 say "Matching records found:"
@ 12, 25 say "Searching record number"
@ 13, 25 say "Total records in file: "
DevOutPict(lastrec(), "#####")
//----- initialize the TBrowse object
oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
oBrowse:headSep := ' '
oBrowse:colSep := ' '
oBrowse:footSep := ' '
oBrowse:skipBlock := { | nRows | gilligan(nRows, oBrowse, @nCurrEle, nMaxEle) }
//----- first column will be locked and shall contain the record number
oBrowse:addColumn( TBColumnNew("Rec #", { || recno() }) )
oBrowse:freeze := 1
//----- init the rest of the columns, one for each field
for x := 1 to fcount()
    oBrowse:addColumn( TBColumnNew(field(x), fieldblock(field(x))) )
next

//----- initialize cargo array to hold record numbers

```

```

//----- initialize cargo array to hold record numbers
oBrowse:cargo := { }
//----- loop through all matching records to build array of record #s
//----- this array will be stored in cargo and used in skipBlock below
do while eval(bCondition) .and. nMaxEle < 4096
  SetPos(11, 51)
  @ 11, 51 say ++nMaxEle picture '####' color '+gr/r'
  aadd(oBrowse:cargo, recno())
  skip
  do while ! eval(bCondition) .and. ! eof() .and. nMaxEle < 4096
    @ 12, 48 say recno() picture '#####'
    skip
  enddo
enddo
if nMaxEle == 4096
  @ 14, 27 say " Limited to 4096 records "
else
  @ 14, 26 say " Press any key to continue "
endif
inkey(0)

go TOPRECORD
//----- modify top/bottom blocks to jump to first/last records in array
oBrowse:goTopBlock := { || dbgoto(TOPRECORD), nCurrEle := 1 }
oBrowse:goBottomBlock := { || dbgoto(BOTTOMRECORD), nCurrEle := nMaxEle }

do while nKey <> K_ESC
  //----- keep cursor out of leftmost column
  if oBrowse:colPos < 2
    oBrowse:colPos := 2
  endif
  dispbegin()
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
  dispend()
  if nKey == 0
    nKey := inkey(0)
  endif
  if nKey == K_ENTER
    if editcell(oBrowse) // returns .T. if record was changed
      //----- test if this record no longer satisfied the filter
      if ! eval(bCondition)
        //----- remove current record from the array

```



```

    adel(oBrowse:cargo, nCurrEle)
    //----- truncate the array
    asize(oBrowse:cargo, len(oBrowse:cargo) - 1)
    nMaxEle--
    //----- prevent array access error if we deleted last element
    nCurrEle := min(nMaxEle, nCurrEle)
    dbgoto(oBrowse:cargo[nCurrEle])
    oBrowse:refreshAll()
  else
    oBrowse:refreshCurrent()
  endif
endif
else
  keytest(nKey, oBrowse)
endif
enddo

//----- restore previous environment
setcursor(nOldcursor)
setcolor(cOldcolor)
restscreen(0, 0, maxrow(), maxcol(), cOldscm)
return nil

static function editcell(oBrowse)
local IRetval := .f.
local oColumn := oBrowse:getColumn(oBrowse:colPos)
local v := eval(oColumn:block)
local nOldcursor := setcursor(SC_NORMAL)
readmodal( { getnew(Row(), Col(), ;
               { | _1 | if(pcount() == 0, v, v := _1) }, ;
               oColumn:heading, '@K', oBrowse:colorSpec) } )
if v <> eval(oColumn:block) // i.e., variable was changed
  if rlock() // make sure record is locked
    eval(oColumn:block, v) // this changes the field
  unlock
endif
IRetval := .t.
endif
setcursor(nOldcursor)
return IRetval

static function gilligan(nRows, oBrowse, nCurrEle, nMaxEle)
local nMovement := 0
do case

```

```
//--- startup mode
case nRows == 0
  skip 0
//--- moving forward
case nRows > 0
  do while nMovement < nRows .and. nCurrEle < nMaxEle
    go oBrowse:cargo[++nCurrEle]
    nMovement++
  enddo

//--- moving backward
case nRows < 0
  do while nMovement > nRows .and. nCurrEle > 1
    go oBrowse:cargo[--nCurrEle]
    nMovement--
  enddo
endcase
return nMovement
```

استخدام "الشحنة" Cargo لمسح حقول حرفية عريضة

قد يكون لدينا حقل حرفي عريض ولريد جزءاً منه في نافذة جدول الاستعراض TBrowse مع الإبقاء على إمكانية مشاهدة الحقل بأكمله. ويكمن الحل في وجود متغير فوري "الشحنة" Cargo لكل عمود ، حيث يقوم هذا المتغير بإيقاف المؤشر مبيناً مكانه في هذا الحقل. عندما ننشئ العمود سنستخدم الوظيفة SUBSTR() بالاشراك مع ذلك المؤشر.

يمكننا ، إذا أردنا التوسع في ذلك بحيث يكون لدينا مصفوفة في حيز الشحنة يتيح لنا حفظ مؤشرات منفصلة لكل سجل. ولكن هذا يعتبر إسرافاً في معظم الحالات.

لاحظ أن الأمر يبدو غير عادي أن نشير إلى العمود من داخل كتلته الخاصة به ولكن هذا منطقي (وضروري في هذه الحالة).

```
c := TBColumnNew( "nameless Truncated field" , ;
  { | | substr(dummy->address, b:getcolumn(2):cargo ) } )
```

قبل إدخال العمود هذا في هدف جدول الاستعراض TBrowse ينبغي القيام بإجراءات أخرى.

أولاً : ينبغي ضبط عرض العمود ، وإلا سيضبطه جدول الاستعراض TBrowse تلقائياً وفق طول الحقل بأكمله.

c:width := 57

يلي ذلك ، ينبغي أن نستهل "الشحنة" cargo بالقيمة 1:

c:cargo := 1

ثانياً : سنضيف "كتلة الألوان" colorBlock التي ستلون البيانات التي مرّ عليها المسح بلون مختلف (راجع التغير الفوري "كتلة الألوان" colorBlock للاطلاع على معلومات أكثر تفصيلاً). وفي هذه الحالة ، لن يعتمد التغير الفوري "كتلة الألوان" colorBlock على البيانات ذاتها ، بل على كون مؤشر المجموعة الفرعية substring للسجل قد تغير أم لا. وبالنسبة لهذا السجل سينتظر التغير الفوري "كتلة الألوان" colorBlock في مصفوفة الشحنة فقط.

c:colorBlock := { | | if(b:getColumn(2):cargo > 1 , ;
{ 3, 4 } , { 1, 2 }) }

وأخيراً ، ينبغي علينا أن نزيد من قوة القاعدة المنطقية لمفتاح الأسهم. سيدقق مفتاحا السهمين الأيسر ← والأيمن → في وجود شحنة رقمية. ويجب علينا في هذه الحالة تحديد موقعنا ضمن البيانات. فمثلاً إذا ضغطنا مفتاح السهم الأيسر ونحن في آخر الحقل الطويل من الجهة اليسرى ، فيجب حينئذٍ تغيير الأعمدة ، وإلا فسيبقى المؤشر في موضعه في العمود إلى مالانهاية.

سيدقق مفتاحا السهمين الأعلى والأسفل (وكذلك الحال في أي انتقال عمودي) أيضاً وجود شحنة رقمية فإن وجدت سنجعل الحقل المسوح يعود "بسرعة" إلى أقصى اليسار بإعادة ضبط مؤشر الشحنة إلى (١). كما ينبغي إجراء "تجديد" بالصف الحالي ليعرض هذا التغير على الشاشة.

```

// filename : TBROW32.PRG
#include "inkey.ch"
#include "box.ch"

#define CURR_COLUMN oBrowse:getColumn(oBrowse:colPos)

function tbrow32
local oBrowse := TBrowseDB(1, 1, maxrow() - 1, maxcol() - 1)
local oColumn
local nKey
local x
local nOldcursor := setcursor(0)
? "creating test database..."
dbcreate("tbrow32", { { "NAME", "C", 20, 0 }, ;
                     { "ADDRESS", "C", 80, 0 } } )
scroll()
use tbrow32
for x := 1 to 50
    append blank
    tbrow32->name := "Record #" + ltrim(str(x))
    tbrow32->address := "This is truncated - use left/right " + ;
                        "arrows to view the rest of this long field!!"
next
go top
oBrowse:colorSpec := "w/b, +w/r, n/bg, +w/bg"
oBrowse:headSep := chr(205)
oColumn := TBColumnNew("Name", { || tbrow32->name } )
oBrowse:AddColumn(oColumn)
oColumn := TBColumnNew("Nameless Truncated Field", ;
                        { || substr(tbrow32->address, ;
                                oBrowse:getColumn(2):cargo ) } )
oColumn:width := 57
oColumn:cargo := 1
//----- show panned fields in a different color
oColumn:colorBlock := { || if(oBrowse:getColumn(2):cargo > 1, ;
                             { 3, 4 }, { 1, 2 } ) }
oBrowse:AddColumn(oColumn)
scroll()
setcolor('w/b')
@ 0, 0, maxrow(), maxcol() box B_SINGLE + ''
do while nKey <> K_ESC .and. nKey <> K_ENTER
    dispbegin()
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
    dispend()
    if nKey == 0

```

```
nKey := inkey(0)
endif
do case

case nKey == K_UP
  if CURR_COLUMN:cargo <> NIL
    oBrowse:refreshCurrent()
    CURR_COLUMN:cargo := 1
  endif
  oBrowse:up()

case nKey == K_DOWN
  if CURR_COLUMN:cargo <> NIL
    oBrowse:refreshCurrent()
    CURR_COLUMN:cargo := 1
  endif
  oBrowse:down()

case nKey == K_LEFT
  if CURR_COLUMN:cargo <> NIL .and. CURR_COLUMN:cargo > 1
    CURR_COLUMN:cargo--
    oBrowse:refreshCurrent()
  else
    oBrowse:left()
  endif

case nKey == K_RIGHT
  if CURR_COLUMN:cargo <> NIL .and. ;
    CURR_COLUMN:cargo < CURR_COLUMN:width
    CURR_COLUMN:cargo++
    oBrowse:refreshCurrent()
  else
    oBrowse:right()
  endif

case nKey == K_HOME
  oBrowse:home()

case nKey == K_END
  oBrowse:end()

case nKey == K_PGUP
  if CURR_COLUMN:cargo <> NIL
    oBrowse:refreshCurrent()
    CURR_COLUMN:cargo := 1
  endif
  oBrowse:pageUp()
```



```
case nKey == K_PGDN
  if CURR_COLUMN:cargo <> NIL
    oBrowse:refreshCurrent()
    CURR_COLUMN:cargo := 1
  endif
  oBrowse:pageDown()
endcase
enddo
use
ferase("tbrow32.dbf")
setcursor(nOldcursor)
return nil
```

استعراض مصادر بيانات بديلة

بعد استيعاب المتغير الفوري "تجاوز الكتلة" b:skipBlock ، يمكننا الانتقال لاستعراض أشياء أخرى غير قواعد البيانات. وفي الواقع ، يمكن استعراض كل ماله بنية منتظمة بواسطة فئة هدف جدول الاستعراض TBrowse.

استعراض مصفوفات بسيطة

عندما نستعرض قواعد البيانات ، ندخل اسم الحقل في تعريف عمود الاستعراض TBColumn. وكلما تحرك مؤشر السجل ، يحدد العمود تلقائياً قيمة الحقل في السجل الجديد. وبما أن المصفوفات ليس لها مؤشر سجل خاص بها ، فعلينا أن ننشئ مؤشراً خاصاً ونشير إليه في تعريف العمود.

```
oColumn := TBColumnNew( "", { | | aArray[ele] } )
```

سيستهل المتغير (nEle) بالقيمة (١) ، ويقوم بدور مؤشر في المصفوفة بحيث يعرف جدول الاستعراض TBrowse دائماً عنصر المصفوفة الحالي. وسيمرر (nEle) بالإشارة إلى وظيفة "تجاوز الكتلة" SkipBlock العادية.

فيما يلي كتلات شيفرة الانتقال الثلاث اللازمة لاستعراض مصفوفة بسيطة:

```
b:skipBlock := { | nskipCnt | gilligan(@nEle, nSkipCnt, nMaxEle) }
b:goTopBlock := { | nEle := 1 }
b:goBottomBlock := { | | nEle := nMaxEle }
```

يكون المتغير (nMaxEle) قد أسس وفق طول المصفوفة ، ويعمل كحد ضمن "تجاوز الكتلة".

استعراض مصفوفات متداخلة

يمكن استخدام طريقة استعراض مصفوفة بسيطة في استعراض مصفوفات متداخلة. الفارق الهام والوحيد هو أنه ينبغي تضمين رمز سفلي subscript إضافي لكل عنصر في المصفوفة المتداخلة عند تحديد أهداف عمود جدول الاستعراض TBColumn:

```
#include "directry.ch"
```

```
oColumn := TBColumnNew("Name", { || aFiles[nEle, F_NAME] } )
oColumn := TBColumnNew("Size", { || aFiles[nEle, F_SIZE] } )
oColumn := TBColumnNew("Date", { || aFiles[nEle, F_DATE] } )
oColumn := TBColumnNew("Time", { || aFiles[nEle, F_TIME] } )
```

فيما يلي شيفرة استعراض المصفوفات المتداخلة. لاحظ معالجة المتغيرات الفورية "لعرض العمود" column:width للتأكد أن العرض كافٍ ، وهذا هام خاصة عند معالجة أسماء الملفات.

```
// filename:TBROW34.PRG
#include "directry.ch"
#include "inkey.ch"

function tbrow34
local aFiles := directory()
local oBrowse := TBrowseNew(0, 10, maxrow(), 69)
local oColumn
local nKey
local nFiles
local nOldcursor := setcursor(0)
local nEle
scroll()
oBrowse:colorSpec := "n/bg, +w/bg, +w/r, +gr/r, +w/rb, +gr/rb, +w*/r"
oBrowse:headSep := chr(205) + chr(209) + chr(205)
oBrowse:colSep := chr(32) + chr(179) + chr(32)
oBrowse:footSep := chr(205) + chr(207) + chr(205)
nEle := 1 // this will serve as our placeholder in the array
nFiles := len(aFiles)
oBrowse:goTopBlock := { || nEle := 1 }
oBrowse:goBottomBlock := { || nEle := nFiles }
oBrowse:skipBlock := { |SkipCnt| gilligan(@nEle, SkipCnt, nFiles) }
oColumn := TBColumnNew("Name", { || aFiles[nEle, F_NAME] })
```

```
//----- highlight .PRG files
oColumn:colorBlock := { | x | if(".PRG" $ x, { 3, 4 }, { 1, 2 }) }
oColumn:width := 12
oBrowse:AddColumn(oColumn)

oColumn := TBColumnNew(" Size", { | | aFiles[nEle, F_SIZE] })
oColumn:width := 10
oBrowse:AddColumn(oColumn)

oColumn := TBColumnNew(" Date", { | | aFiles[nEle, F_DATE] })
//----- highlight files with date stamp matching system date
oColumn:colorBlock := { | x | if(x == date(), { 5, 6 }, { 1, 2 }) }
oColumn:width := 8
oBrowse:AddColumn(oColumn)

oColumn := TBColumnNew(" Time", { | | aFiles[nEle, F_TIME] })
//----- highlight files created between 3 a.m. and 4 a.m. for
//----- the simple reason that you shouldn't be working then!
oColumn:colorBlock := { | x | if(substr(x, 1, 2) $ "03 04", ;
                                { 7, 4 }, { 1, 2 }) }
oColumn:width := 8
oBrowse:AddColumn(oColumn)

do while nKey <> K_ESC .and. nKey <> K_ENTER
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  keytest(nKey, oBrowse)
enddo
setcursor(nOldcursor)
return aFiles[nEle, F_NAME]

static function gilligan(nEle, nSkipCnt, nMaxEle)
local nOldEle := nEle
//----- moved past bottom
if nEle + nSkipCnt > nMaxEle
  nEle := nMaxEle
//----- moved past top
elseif nEle + nSkipCnt < 1
  nEle := 1
else
```

```

nEle += nSkipCnt
endif
return nEle - nOldEle

```

استعراض مصفوفات متداخلة بطول غير معروف

إن كان طول المصفوفات المتداخلة غير معروف لنا قبل عملية التشغيل ، فالحل المباشر هو حلقة "FOR...NEXT" مثل:

```

for x := 1 to len(array[1])
  oBrowse.addColumn( TBColumnNew( , { | | array[nEle, x] } ) )
next

```

لكن هذا ليس حلاً لأن الإشارة إلى عداد الحلقة (x) غير "مصممة" (أو مشفرة في البرنامج) ضمن كتلة الشيفرة. ولذلك ، حالما نحاول وظيفة "التأسيس" (stabilize) تقيّم البيانات التي في هذه الأعمدة سيتدمر برنامجك لأن (x) ستكون أكبر من أقصى عدد من العناصر في كل مصفوفة متداخلة (وهذا إجراء عادي بالنسبة لعداد الحلقة بعد انتهاء الحلقة).

يمكن في هذه الحالة استخدام "متغيرات عملية مستقلة" . فعندما تعيد إحدى الوظائف كتلة شيفرة تشير إلى متغير محلي بالنسبة إلى تلك الوظيفة ، تبقى هذه المتغيرات المحلية مادامت كتلة الشيفرة تشير إليها. ولذلك يمكنك استدعاء الوظيفة ذاتها عشر مرات لإنشاء عشر حوادث ظهور مختلفة للمتغير المحلي ذاته. إن المتغيرات المحلية المستقلة حل مثالي للإشارة إلى عدادات الحلقات ضمن كتل الشيفرة.

إن المثال التالي هو مصفوفة استعراض متداخلة عامة تستخدم فيها المتغيرات المحلية المستقلة. لاحظ أن مؤشر (nEle) أصبح الآن ثابتاً على عرض الملف ، وهذا ضروري لأنه يجب أن يكون مرئياً في "تكوين الكتلة" MakeBlock الذي يجب أن يشير إليه ضمن كتلة الشيفرة لكل عمود. تستدعي هذه الوظيفة مرتين: مرة مع

المصفوفات المتداخلة الرقمية ، وأخرى مع مصفوفة وظيفة "الدليل"
DIRECTORY()

ومع أن للمصفوفات المتداخلة ضمن هذه المصفوفات أطوال مختلفة ، تغير
الشفرة ذاتها بداتها لمعالجتها دون أي خطأ ، وهذا بفضل وجود المتغيرات المحلية
المستقلة.

يبين هذا المثال أيضاً كيفية اعداد استعراض مصفوفة ، ويتيح تعديل أو تحرير
عناصر مصفوفة معينة في مكانها. تنشئ وظيفة "تكوين الكتلة" MakeBlock كتلة
شفرة استرجاعية / تعينه (أو أوامر "get/set") لها تركيب ملائم لهدف أوامر GET.
(لاحظ أن هذا أيضاً بنية تامة لكتلة الشفرة راجعة بواسطة وظيفتي "كتلة الحقل"
FIELDWBLOCK() و FIELDBLOCK().

```
// filename: TBROW35.PRG
#include "setcurs.ch"
#include "inkey.ch"

static nEle := 1 // this will serve as our placeholder in the array

#define TEST // to compile test stub

#ifdef TEST

function tbrow35
//----- try it first with nested arrays of three elements each
browseit( { { 1, 2, 3 }, ;
           { 4, 5, 6 }, ;
           { 7, 8, 9 }, ;
           {10,11,12 }, ;
           {13,14,15 }, ;
           {16,17,18 }, ;
           {19,20,21 } } )
//----- then again with a DIRECTORY() array, which contain 5 elements each
browseit(directory())
return nil

#endif
```

```

static function browseit(aData)
local x
local oColumn
local nKey
local nMaxEle := len(aData)
local nOldcursor := setcursor(0)
local oBrowse := TBrowseNew(0, 0, maxrow(), maxcol())
local aMorekeys := { { K_ENTER, { | b | editcell(b) } } }
set scoreboard off
scroll()
oBrowse:colorSpec := "n/bg, +w/bg"
oBrowse:headSep := chr(205) + chr(209) + chr(205)
oBrowse:colSep := chr(32) + chr(179) + chr(32)
oBrowse:footSep := chr(205) + chr(207) + chr(205)
oBrowse:goTopBlock := { || nEle := 1 }
oBrowse:goBottomBlock := { || nEle := nMaxEle }
oBrowse:skipBlock := { | nSkipCnt | gilligan(nSkipCnt, nMaxEle) }
for x := 1 to len(aData[1])
  oColumn := TBColumnNew(, makeblock(aData, x))
  //----- next line is for safety... delete if you don't need it
  oColumn:width := 12
  oBrowse:AddColumn(oColumn)
next
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  keytest(nKey, oBrowse, aMorekeys)
enddo
setcursor(nOldcursor)
return nil

```

```

/*
  Function: MakeBlock()
  Purpose: Use detached locals to resolve loop counter reference
*/

```

```

static function makeblock(a, e)
return { | _1 | if(_1 == NIL, a[nEle, e], a[nEle, e] := _1) }

```

```

static function gilligan(skip_cnt, maxval)
local nOldEle := nEle
//----- moved past bottom

```

```

if nEle + skip_cnt > maxval
    nEle := maxval
//----- moved past top
elseif nEle + skip_cnt < 1
    nEle := 1
else
    nEle += skip_cnt
endif
return nEle - nOldEle

static function editcell(oBrowse)
local oColumn := oBrowse:getColumn(oBrowse:colPos)
local nOldcursor := setcursor(SC_NORMAL)
readmodal( { getnew(Row(), Col(), oColumn:block, oColumn:heading, ;
    '@K', oBrowse:colorSpec) } )
setcursor(nOldcursor)
if lastkey() <> NIL
    oBrowse:refreshCurrent()
endif
return nil

```

استعراض حقول المذكرة Memo Fields

يمكن هنا أيضاً استخدام طريقة استعراض مصفوفة بسيط لاستعراض حقول المذكرة ، والفارق هو محتويات هدف عمود جدول الاستعراض TBColumn ، والحد الأقصى للعناصر في "الانتقال إلى كتلة أسفل" goBottomBlock :

```

column := TBColumnNew( " " , | | memoline(mfile, 80, ele) } )
max_ele := mlcount(mfile, 80)

```

ينقل المثال التالي ملف نص إلى المذكرة باستخدام وظيفة "قراءة المذكرة" MEMOREAD() التي تحاكي "التحرك-الإقفال" scroll-Lock بواسطة مرشح الصف مباشرة عند ضغط مفتاح السهم "أعلى" أو "أسفل" الذي يجعل الاستعراض Tbrowse يحرك البيانات إلى أعلى وإلى أسفل.

```

// filename:TBROW36.PRG
#include "inkey.ch"

function tthrow36(cFile)
local cOldscrm := savescreen(0, 0, maxrow(), maxcol())
local oBrowse := TBrowseNew(0, 0, maxrow(), 79)
local nKey
local nEle
local nMaxEle
local cMemo
local nOldcursor := setcursor(0)
local nVWidth := maxcol() + 1
if cFile == NIL
    ? "Syntax: MEMOBROW <cFile>"
    quit
endif
cMemo := memoread(cFile)
nEle := 1 // this will serve as our placeholder in the memo
nMaxEle := mlcount(cMemo, nVWidth)
oBrowse:colorSpec := "+w/b, +w/b"
oBrowse:skipBlock := { | nSkipCnt | gilligan(@nEle, nSkipCnt, nMaxEle) }
oBrowse:goTopBlock := { || nEle := 1 }
oBrowse:goBottomBlock := { || nEle := nMaxEle }
oBrowse:AddColumn( TBColumnNew(, { || memoline(cMemo, nVWidth, nEle) }) )
do while nKey <> K_ESC .and. nKey <> K_ENTER
    dispbegin()
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    dispend()
    if nKey == 0
        nKey := inkey(0)
    endif
    do case
        case nKey == K_UP
            // force TBrowse to refresh entire screen
            oBrowse:rowPos := 1
            oBrowse:up()
        case nKey == K_DOWN
            // force TBrowse to refresh entire screen
            oBrowse:rowPos := oBrowse:rowcount
            oBrowse:down()
        case nKey == K_PGUP
            oBrowse:pageUp()
        case nKey == K_PGDN
            oBrowse:pageDown()
        endcase
    enddo
enddo

```

```
setcursor(nOldcursor)
restscreen(0, 0, maxrow(), maxcol(), cOldscm)
return nil

static function gilligan(nEle, nSkipCnt, nMaxEle)
local nOldEle := nEle
//----- moved past bottom
if nEle + nSkipCnt > nMaxEle
    nEle := nMaxEle
//----- moved past top
elseif nEle + nSkipCnt < 1
    nEle := 1
else
    nEle += nSkipCnt
endif
return nEle - nOldEle
```

ليس هذا هو الحل الأمثل ، لأن ملفك محدد بحجم أقصى قدره ٦٤ ك (64k) (وهو حد المعيار القديم لكليبر الخاص بحجم السلسلة الحرفية). ومع ذلك ، يتضمن الأسطوانة المرن لشيفرة المصدر ملف TEXTBROW.PRG الذي يحتوي وظيفة "استعراض النص" TextBrow() التي تتيح لك استعراض ملفات محددة من النصوص. كما يمكنك البحث عن كلمات ضمن الملفات ، وتظليل أقسام من الملف لطباعتها أو نسخها إلى ملف آخر. يتطلب "استعراض النص" TextBrow() استخدام العديد من وظائف سي ، المتضمنة في الملف FTTEXT.OBJ لتشاهد وظيفة "استعراض النص" TextBrow() فعلياً ، اطبع RMAKE TEXTBROW.

استعراض ملفات Btrieve

إن ملفات Btrieve لها بنية منتظمة ، لذلك يمكن أيضاً استعراضها بواسطة جدول الاستعراض TBrowse.

تختلف طريقة تجاوز السجلات في Btrieve عنها في كليبر. فإن Btrieve يتجاوز فقط سجلاً واحداً في كل مرة ، كما يجب عليك إخباره بجهة الانتقال. ولتجاوز سجلات متعددة ، يجب استخدام حلقة. لذلك تم تشكيل "تجاوز الكتلة" skipBlock لتمرير أعلى عدد من الصفوف المطلوب تجاوزها في وظيفة () Gilligan ، إضافة إلى قيمة منطقية تدل على الاتجاه في الملف ، إلى الأمام أو الخلف (تعني "T." الاتجاه إلى الأمام).

يعتمد المثال التالي على ملف الزويسة DBFTRVE.CH ، ويجب ربطه بملف المكتبة DBFTRVE.LIB. ويوجد هذان الملفان في المكتبة DBFtrieve التي توفر ارتباطاً وثيقاً بين كليبر و Btrieve. وإذا أردت العمل على ملفات Btrieve فأفضل اختيار هو DBFtrieve .

```
// filename: TBROW37.PRG
#include "dbftrve.ch"
#include "inkey.ch"
#include "setcurs.ch"
#include "box.ch"

function tbrow37
local aBP := B_Init( 28, 8, 0 )      // Set up Btrieve param array
local oBrowse := TBrowseNew(4, 24, 20, 55) // Create the browse object
local nKey := 0                      // Keystroke container

setcursor(0)

//----- Paint the initial screen
@ 1, 0, maxrow(), maxcol() box replicate(chr(178), 9) color "BG+/B"
@ 3, 23, 21, 56 box B_DOUBLE + " " color "W+/B"

@ 5, 23 say " " color "W+/B"
@ 5, 56 say " " color "W+/B"
@ 0, 0, 0, 79 box space(9) color "R/R"
@ 0, 29 say " Btrieve TBrowse Demo " color "G+/N"

// Add two columns to the browse object. The first column contains the
// name, which is contained in the first 20 bytes of the record buffer
// (the record buffer is element #1 of the Btrieve param array). The
// second column contains the IQ. It is stored in the Btrieve file as
// an IEEE 64-bit float, so the B_Float8() function is used to handle
// the conversion.
```

```

oBrowse:addColumn( TBColumnNew( " Speaker", ;
    {|| " " + substr( aBP[ 1 ], 1, 20 )} ) )
oBrowse:addColumn( TBColumnNew( " IQ ", ;
    {|| transform( B_Float8( substr( aBP[ 1 ], 21, 8 ) ), ;
    "9999" ) + " " } ) )

// Set miscellaneous instance variables
oBrowse:autoLite      := .F. // because we will highlight the entire row
oBrowse:colorSpec     := "W+/B,GR+/RB"
oBrowse:colSep        := chr(32)+chr(179)+chr(932)
oBrowse:headSep       := replicate(chr(196), 3)

// Set up data retrieval blocks. Note use of custom skip block.

oBrowse:goBottomBlock := {|| B_GoBottom( aBP )}
oBrowse:goTopBlock    := {|| B_GoTop( aBP )}
oBrowse:skipBlock     := {|| nRecs | Gilligan( aBP, abs( nRecs ), nRecs > 0 )}

// Create a Btrieve file for use by the demo. Very simple, it contains
// only name and IQ, and is keyed in ascending order by IQ.
B_Create( "TEST.DTA", 28, 512, 1, .F., .F., .F., .F., .F., .F., .F., 0, ;
    { 21 }, { 8 }, { .T. }, { .F. }, { .F. }, { .F. }, { .F. }, ;
    { .F. }, { .F. }, { .T. }, { .F. }, { 2 }, { chr( 0 ) } )

// Open the Btrieve file
B_Use( aBP, "TEST.DTA" )

// Add some records. Note use of C_Float8() to handle conversion from
// Clipper numeric to IEEE 64-bit float.

B_Append( aBP, "Ted Means      " + C_Float8( 197 ) )
B_Append( aBP, "Greg Lief     " + C_Float8( 88 ) )
B_Append( aBP, "Ira Emus      " + C_Float8( 12 ) )
B_Append( aBP, "Mark Worthen  " + C_Float8( 45 ) )
B_Append( aBP, "Justin Lief   " + C_Float8( 209 ) )
B_Append( aBP, "Jennifer Lief " + C_Float8( 211 ) )

//----- Go to logical top-of-file
B_GoTop( aBP )

//----- main event loop
do while nKey <> K_ESC

    dispbegin()
    do while ! oBrowse:stabilize()
    enddo

```

```
enddo
dispend()

//----- highlight entire row
oBrowse:colorRect( {oBrowse:rowPos, 1, oBrowse:rowPos, 2}, {2, 1} )
nKey := inkey( 0 )
//----- dehighlight entire row
oBrowse:colorRect( {oBrowse:rowPos, 1, oBrowse:rowPos, 2}, {1, 2} )
do case
  case nKey == K_DOWN
    oBrowse:down()
  case nKey == K_UP
    oBrowse:up()
  case nKey == K_HOME
    oBrowse:rowPos := 1
    oBrowse:refreshAll()
  case nKey == K_END
    oBrowse:rowPos := oBrowse:rowCount
    oBrowse:refreshAll()
  case nKey == K_PGUP
    oBrowse:pageUp()
  case nKey == K_PGDN
    oBrowse:pageDown()
  case nKey == K_CTRL_PGUP
    oBrowse:goTop()
  case nKey == K_CTRL_PGDN
    oBrowse:goBottom()
endcase
enddo
B_Close( aBP )
return NIL

//----- Custom skipblock function
static function Gilligan( aBP, nRecs, IForward )
local nCount := 0 // Counts number of records skipped

// Loop until desired number of records skipped, or B_Skip() returns
// an error condition, typically bof() or eof().
do while nCount < nRecs .and. B_Skip( aBP, IForward ) == 0
  nCount++
enddo
return if( IForward, nCount, 0 - nCount )
```

جداول استعراضات متعددة متزامنة

كان من الممكن وجود عدة وظائف DBEDIT() في كليبز "Summer'87" بوقت واحد. ومن السهل تطبيق ذلك في جدول الاستعراض TBrowse. ويمكنك أيضاً ربط جداول الاستعراضات ببعضها بحيث يتم تحديث الاستعراضات الأخرى في الوقت الذي تتحرك فيه خلال إحدى النوافذ. ويبين المثال التالي كيفية القيام بذلك.

تقبل وظيفة "جدول استعراضات متعددة" (MultBrow) اسم قاعدة البيانات كمتغير وتفتحها ، وتنشئ تسعة أهداف استعراض TBrowse. ويعرض الاستعراض الأول رقم السجل ويعرض كل واحد من بقية الاستعراضات حقل قاعدة البيانات. وإذا كان عدد حقول قاعدة البيانات أقل من ثمانية ، يبدأ البرنامج ثانية في الحقل الأول.

لقد تم تهيئة مفاتيح [Tab] و [Shift]-[Tab] للانتقال بين النوافذ. أما مفاتيح الأسهم إلى الأعلى وإلى الأسفل فينقلانك عبر قاعدة البيانات. لاحظ أنه يمكن تمييز النافذة النشطة الحالية بسهولة بواسطة إطارها السميك. وستلاحظ أيضاً أن لون عمود التظليل يتغير كلما انتقلت من نافذة إلى أخرى. ويتم ذلك بشكل جزئي بمعالجة المتغير الفوري "طيف الألوان" b:colorSpec .

يحتوي هذا المثال جزئين هامين ، أولهما : فحص القاعدة المنطقية لحالة مفاتيح [Tab] و [Shift]-[Tab]. وليس ذلك بتغيير "طيف الألوان" b:colorSpec فحسب ، بل إننا كلما غيرنا متغيرات فورية (باستثناء b:cargo) علينا استدعاء وظيفة "التشكيل" (b:configure) لإعادة تشكيل هدف جدول الاستعراض ، إضافة إلى ذلك ، يجب استدعاء وظيفة "التأسيس" (b:stabilize) فبدونها لا يكون لوظيفة "تجديد الكل" (b:refreshAll) أي تأثير.

بعد إجراء هذه الخطوات على هدف جدول الاستعراض TBrowse الحالي نتقل بعدها إلى الهدف الجديد ونعيد العملية ذاتها. لاحظ أنه لا حاجة لتضمين عملية

استدعاء الوظيفة "التأسيس" (b:stabilize) ثانياً ، لأنه سيمر معنا في أعلى حلقة "DO WHILE".

بعد حلقة التأسيس مباشرة يظهر أمامنا الجزء الثاني الذي "يربط" نوافذ الاستعراض مع بعضها. تطبق وظيفة "تجديد الكل" (refreshAll) على كافة الاستعراضات (ماعدًا النشاط حالياً) ثم يتم تثبيتها. لاحظ أيضاً استخدام وظيفتي "بداية العرض" (DISPBEGIN) و "نهاية العرض" (DISPEND) أثناء هذه العملية. وهذا يضمن عرض جميع التحديثات بوقت واحد.

(لاحظ أيضاً استخدام "المتغيرات المحلية المستقلة" "delached locals" في وظيفة "تكوين كتلة" (MakeBlock) لإنشاء كتل الأعمدة. إنها ضرورية لتشفير الإشارة إلى عداد حلقة "FOR...NEXT" في البرنامج).

```
//filename : TBROW38.PRG
#include "inkey.ch"
#include "box.ch"

#define ACTIVE_COLORS 'W/B,+GR/R,N/N,N/N,N/W'
#define INACTIVE_COLORS 'W/B,+W/RB,N/N,N/N,N/W'
#define TOP 1
#define LEFT 2
#define BOTTOM 3
#define RIGHT 4

#define BROWSES 9

function tbrow38(cDbfname)
local aBrowses := array(BROWSES)
local x
local oColumn
local nCurrBrow
local nFields
local aCoords := { { 1, 1, 5, 24 }, ;
                    { 3, 28, 7, 51 }, ;
                    { 1, 55, 5, 78 }, ;
                    { 9, 55, 13, 78 }, ;
                    { 11, 28, 15, 51 }, ;
                    { 17, 55, 21, 78 }, ;
                    { 19, 28, 23, 51 }, ;
```



```

                {17, 1, 21, 24}, ;
                { 9, 1, 13, 24}}
local nOldcursor := setcursor(0)
if cDbfname <> NIL
    scroll()
    setcolor('W/N')
    use (cDbfname) new
    nFields := fcount()
    dispbegin()
    for x := 1 to BROWSES
        aBrowses[x] := TBrowseDB(aCoords[x, TOP], aCoords[x, LEFT], ;
                                aCoords[x, BOTTOM], aCoords[x, RIGHT])
        if x == 1
            aBrowses[x]:colorSpec := ACTIVE_COLORS
        else
            aBrowses[x]:colorSpec := INACTIVE_COLORS
        endif
        aBrowses[x]:headSep := ' '
        if x == BROWSES
            oColumn := TBColumnNew("Rec #", { || recno() } )
        else
            oColumn := TBColumnNew(field((x - 1) % nFields + 1), ;
                                    makeblock(x, nFields))
        endif
        aBrowses[x]:AddColumn(oColumn)
        @ aBrowses[x]:nTop - 1, aBrowses[x]:nLeft - 1, ;
        aBrowses[x]:nBottom + 1, aBrowses[x]:nRight + 1 ;
        box B_DOUBLE + ' ' color 'W/B'
        do while ! aBrowses[x]:stabilize()
            enddo
        next
        nCurrBrow := 1
        dispend()
        do while lastkey() <> K_ESC
            @ aBrowses[nCurrBrow]:nTop - 1, ;
            aBrowses[nCurrBrow]:nLeft - 1, ;
            aBrowses[nCurrBrow]:nBottom + 1, ;
            aBrowses[nCurrBrow]:nRight + 1 box B_DOUBLE color 'W/B'
            browseit(aBrowses, @nCurrBrow)
        enddo
        use
    endif
    setcursor(nOldcursor)
    return nil

// end function Main()
//-----//

```

```
//-----//
```

```
function makeblock(num, nFields)
return { || fieldget((num - 1) % nFields + 1) }
```

```
// end static function MakeBlock()
//-----//
```

```
function Browselt(aBrowses, nPtr)
local oBrowse := aBrowses[nPtr]
local nKey
local nTempRow
local x
do while nKey <> K_ESC .and. nKey <> K_TAB .and. nKey <> K_SH_TAB
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    //----- restabilize all other browses
    dispbegin()
    nTempRow := oBrowse:rowPos
    for x := 1 to BROWSES
      if x <> nPtr
        aBrowses[x]:rowPos := nTempRow
        aBrowses[x]:refreshAll()
        do while ! aBrowses[x]:stabilize()
        enddo
      endif
    next
    dispend()
    nKey := inkey(0)
  endif
  if nKey == K_TAB .or. nKey == K_SH_TAB
    //----- change colorSpec of active browse and redisplay
    @ oBrowse:nTop - 1, oBrowse:nLeft - 1, oBrowse:nBottom + 1, ;
      oBrowse:nRight + 1 box B_DOUBLE + ' ' color 'W/B'
    oBrowse:colorSpec := INACTIVE_COLORS
    oBrowse:invalidate()
    dispbegin()
    do while ! oBrowse:stabilize()
    enddo
    dispend()

    //----- change current browse nPtr
```

```
if nKey == K_TAB // moving forward
  if nPtr == BROWSES
    nPtr := 1
  else
    nPtr++
  endif
else // moving backward
  if nPtr == 1
    nPtr := BROWSES
  else
    nPtr--
  endif
endif

//----- change colorSpec of new browse and force redisplay
//----- when we re-enter this function
aBrowses[nPtr]:colorSpec := ACTIVE_COLORS
aBrowses[nPtr]:invalidate()

else
  keytest(nKey, oBrowse)
endif
enddo
return nil
```

الاستعراضات الرئيسية/الفرعية

إحدى المشاكل القديمة في كليب و dBASE هي كيفية عرض نافذة رئيسة (أو ترويسة) إضافة إلى نافذة أخرى تبين كافة السجلات الفرعية (أو التفصيلية) التي تنطبق على كل ترويسة مظلمة.

مرّ معنا أعلاه على نوافذ الاستعراض المتزامنة. السمة الهامة فيها هي شيفرة "متزامنة" النوافذ المتنوعة ، والتي يمكن تطبيقها على نوافذ استعراض السجلات الرئيسية والفرعية.

ننشئ أولاً كتل الانتقال. وبما أننا نتعامل مع منطقتي عمل منفصلتين ، يجب علينا أن نشير إلى النسخة المكافئة لمنطقة عمل السجلات الفرعية ضمن الكتل. فمثلاً:

```
b:goTopBlock      := { | | (child)->(dbseek(startval, .t. )) }
b:goBottomBlock := { | | (child)->(dbseek(startval+1, .t. )), dbskip(-1) }
```

وهذا أفضل من التالي:

```
b:goTopBlock      := { | | dbseek(startval, .t. ) }
b:goBottomBlock := { | | dbseek(endval+1, .t. ), dbskip(-1) }
```

وبهذا لا يكون هناك حاجة لاختيار SELECT منطقة عمل السجلات الفرعية عندما يستلزم الأمر استخدام وظيفة "التثبيت" (stabilize) لهدف الاستعراض ذلك.

سنقوم أيضاً بتخزين "كتلة شيفرة الإشارة في الفهرس" في حيز الشحنة لهذا الاستعراض. وعند تقييمها ، ستعيد كتلة الشيفرة هذه القيمة الحالية لدليل فهرس التحكم لقاعدة البيانات الفرعية. وهذا ضروري في وظيفة التجاوز ، وكذلك التأكد من وجود سجلات فرعية تتوافق مع السجلات الرئيسة الحالية.

لا يتطلب هدف استعراض قاعدة البيانات الرئيسة (أو الترويسة) كتل انتقال خاصة. ومع ذلك ، ينبغي ربط الاستعراض الفرعي بالرئيسي. أحد الحلول الجيدة هو إدخال كتلة شيفرة في حيز شحنة الاستعراض ، وتقييم هذه الكتلة بما يلي:

أ) تأسيس قيمتي البداية والنهاية للمجموعة الجزئية من البيانات (الموجودة في القيم الثابتة STARTVAL و ENDVAL على عرض الملف).

ب) عرض جدول الاستعراض الفرعي وفق القيمة الحالية في النافذة الرئيسة.

```
browes_[1]:cargo := { { | | startval := endval := authors->name, ;
                        showchild(browes_[2] ) } }
```

ملاحظة

لقد جعلنا b:cargo مصفوفة بعنصر واحد للتمييز بين كتل شيفرة الشحنة الخاصة بالاستعراض الرئيسي والاستعراض الفرعي. ولن يكون هذا الإجراء ضرورياً في حال كون الاستعراضات الرئيسية والفرعية مرتبطتين تماماً ، لأننا يمكننا ضبط متغيرين فوريين منفصلين مثل "عرض السجلات الفرعية" showChild و "كتلة الفهرسة" indexBlock.

إن أفضل زمان ومكان لعرض النافذة الفرعية هو بعد تثبيت النافذة الرئيسة مباشرة ، وقبل ضغط المفاتيح. لهذا سنتأكد من وجود مصفوفة شحنة قبل ضغط المفاتيح. فإن وجد سيتم تقييم كتلة الشيفرة التي ستؤدي إلى تجديد نافذة الاستعراض الفرعي حسبما تم تظليله في النافذة الرئيسة.

ستنفذ وظيفة "عرض السجلات الفرعية" (showChild) وظيفة "الانتقال إلى أعلى" (goTop) للاستعراض الفرعي الذي سيبحث في محتويات القيمة الثابتة STARTVAL على عرض الملف. وبعد ذلك يجب فحص "كتلة شيفرة الإشارة في دليل الفهرس" (المخزنة في حيز الشحنة للاستعراض الفرعي) للتأكد من وجود سجلات صحيحة تتوافق مع الرئيسة. فإذا وجدت ، نستخدم حلقة "التثبيت" (stsbilize) لعرض هذه السجلات على الشاشة ، وإلا نفرغ نافذة الاستعراض الفرعي (التي نعرف إحداثياتها من المتغيرات الفورية للاستعراض) ونعرض رسالة في وسط هذه النافذة. (لاحظ أن عمل وظيفة "عرض السجلات الفرعية" showChild يتركز ضمن بنية وظيفة "بداية العرض" (DISPBEGIN) ووظيفة "نهاية العرض" (DISPEND) لمصفوفة عرض البيانات على الشاشة).

وكما في المثال السابق ، يتيح لك مفتاح [Tab] الانتقال بين النوافذ. وعندما تكون في نافذة الاستعراض الفرعي ، يمكنك ضغط مفتاح "إدراج" [Ins] أو مفتاح "حذف" [Del] لإدراج أو حذف سجلات. ولضمان احتواء أي من السجلات الجديدة

لقيمة الدليل ذاتها التي في السجل المظلل حالياً في النافذة الرئيسة ، أضفنا قاعدة منطقية.

```
// filename: TBROW39.PRG
#include "box.ch"
#include "inkey.ch"
#include "setcurs.ch"

#define B_THICK chr(219)+chr(223)+chr(219)+chr(219)+;
               chr(219)+chr(220)+chr(219)+chr(219)

//----- these delineate structure of the TBrowse cargo array
#define _INDEXBLOCK 1
#define _ALIAS      2

static startval, endval

#define TEST // to compile test stub

#ifdef TEST

function tbrow39
setcursor(0)
set deleted on
? "Creating test index"
use articles new

//----- I'm using dbCreateIndex() instead of INDEX ON command so that we
//----- can show indexing progress via ShowDots() function below
dbcreateindex('temp', "articles->name", { || showdots(), articles->name })

use authors new
parentchild('authors','name','articles')
close data
ferase('temp.ntx')
scroll()
return nil

/*
Function: ShowDots()
Purpose: Show status for indexing process
*/
static function showdots
outstd('.')

```

return nil

#endif // end test stub

/*

Function: ParentChild(<cParent>, <cParentkey>, <cChild>)

Purpose: Set up parent and child TBrowse objects

Parameters: <cParent> -- alias of parent work area (already opened)

<cParentkey> -- name of parent field used as link to child

<cChild> -- alias of child work area (already opened with appropriate controlling index)

Returns: Hey mon, no deposit, no return

*/

static function parentchild(cParent, cParentkey, cChild)

local x

local nFields

local nCurrBrow

local aBrowses

setcolor('+w/b,n/bg')

aBrowses := { TBrowseDB(1, 1, 8, maxcol()-1), ;

TBrowseDB(11, 1, maxrow()-1, maxcol()-1) }

aBrowses[1]:addColumn(tbcolumnew(, { || (cParent)-> ;

(fieldget(fieldpos(cParentkey)))) })

aBrowses[1]:cargo := { { || startval := endval := ;

(cParent)->(fieldget(fieldpos(cParentkey))), ;

showchild(aBrowses[2]), cParent }

aBrowses[2]:goTopBlock := { || (cChild)->(dbseek(startval, .t.)) }

aBrowses[2]:goBottomBlock := { || (cChild)->(;

dbseek(substr(endval, 1, len(endval) - 1) + ;

chr(asc(right(endval, 1)) + 1), .t.), ;

(cChild)->(dbskip(-1)) }

aBrowses[2]:cargo := { &("{ || " + cChild + "->(" + ;

(cChild)->(indexkey(0)) + ") }"), cChild }

aBrowses[2]:skipBlock := { | SkipCnt | gilligan(SkipCnt, ;

aBrowses[2]:cargo) }

//----- this example uses all fields in the child database

//----- except for the key field (which we assume to have the

//----- same name as the key field in the parent database)

nFields := (cChild)->(fcount())

for x := 1 to nFields

if upper(cParentkey) <> (cChild)->(field(x))

aBrowses[2]:addColumn(TBColumnNew(, ;

fieldwblock((cChild)->(field(x)), select(cChild))))

```

    endif
next
nCurrBrow := 1
dispbegin()
scroll()
for x := 1 to 2
    @ aBrowses[x]:nTop - 1, aBrowses[x]:nLeft - 1, ;
    aBrowses[x]:nBottom + 1, aBrowses[x]:nRight + 1 box B_SINGLE
next
dispend()
select select(cParent)
do while lastkey() <> K_ESC
    @ aBrowses[nCurrBrow]:nTop - 1, aBrowses[nCurrBrow]:nLeft - 1, ;
    aBrowses[nCurrBrow]:nBottom + 1, aBrowses[nCurrBrow]:nRight + 1 ;
    box B_THICK
    dobrowse(aBrowses, @nCurrBrow)
enddo
return nil

```

```

/*
Function: DoBrowse( <aBrowses>, <nPointer> )
Purpose: Browse display/keystroke workhorse
Parameters: <aBrowses> – array containing parent & child browse
            <nPointer> – # of current browse in <aBrowses>
Returns: Nada
*/

```

```

static function dobrowse(aBrowses, nPtr)
local nKey
local x
local nTemp
local oBrowse := aBrowses[nPtr]
local lCanSwap
do while nKey <> K_ESC .and. nKey <> K_TAB
    do while (nKey := inkey()) == 0 .and. ! oBrowse:stabilize()
    enddo
    lCanSwap := .t.
    if nKey == 0
        if nPtr == 1
            lCanSwap := eval(oBrowse:cargo[1])
        endif
        nKey := inkey(0)
    endif
do case
    case nKey == K_UP
        oBrowse:up()
        oBrowse:stabilize()

```

```
if oBrowse:hitTop
  alert("Top of data")
endif
case nKey == K_DOWN
  oBrowse:down()
  oBrowse:stabilize()
  if oBrowse:hitBottom
    alert("Bottom of data")
  endif
case nKey == K_PGUP
  oBrowse:pageUp()
case nKey == K_PGDN
  oBrowse:pageDown()
case nKey == K_LEFT
  oBrowse:left()
case nKey == K_RIGHT
  oBrowse:right()
case nKey == K_TAB .and. ICanSwap
  @ oBrowse:nTop - 1, oBrowse:nLeft - 1, ;
  oBrowse:nBottom + 1, oBrowse:nRight + 1 box B_SINGLE
  nPtr := if(nPtr == 1, 2, 1)
//----- edit current cell (not applicable in parent area)
case ( nKey == K_ENTER .or. (nKey >= 32 .and. nKey <= 255) ) ;
  .and. nPtr <> 1
  if nKey <> K_ENTER
    keyboard chr(nKey)
  endif
  if editcell(oBrowse)
    oBrowse:refreshCurrent()
  endif
//----- add a new record (not applicable in parent area)
case nKey == K_INS .and. nPtr <> 1
  (oBrowse:cargo[_ALIAS])->( dbappend() )
  if ! empty( x := (oBrowse:cargo[_ALIAS])->( indexkey(0) ) )
    if ( nTemp := at('-', x) ) > 0
      x := substr(x, nTemp + 2)
    endif
    (oBrowse:cargo[_ALIAS])->( fieldput( fieldpos(x), startval) )
  endif
  oBrowse:refreshAll()
//----- delete current record (not applicable in parent area)
case nKey == K_DEL .and. nPtr <> 1
  if (oBrowse:cargo[_ALIAS])->( rlock() )
    (oBrowse:cargo[_ALIAS])->( dbdelete() )
    (oBrowse:cargo[_ALIAS])->( dbunlock() )
```



```

        oBrowse:refreshAll()
    endif
endcase
enddo
return nil

/*
Function: ShowChild( <oBrowse> )
Purpose:  Display the child browse without key processing
          Synchronizes the child window to update in
          accordance with changes in the parent window
Parameter: <oBrowse> = child TBrowse object
Returns:  Nuthin' worth writing home about
*/
static function showchild(oBrowse)
local keyval
local nOldRow := row()
local nOldCol := col()
local nMidRow
local nMidCol
local lRetVal := .t.
local x := 0
dispbegin()
oBrowse:goTop()
keyval := eval(oBrowse:cargo[1])
//----- make sure there are actually valid child records
if startval <= keyval .and. endval >= keyval
do while eval(oBrowse:skipBlock, 1) <> 0 .and. x++ < oBrowse:rowCount
enddo
oBrowse:goTop()
oBrowse:refreshAll()
do while ! oBrowse:stabilize()
enddo
if x > oBrowse:rowCount
    @ oBrowse:nBottom + 1, oBrowse:nLeft + 2 say "More " + chr(25)
else
    @ oBrowse:nBottom + 1, oBrowse:nLeft + 2 say replicate(chr(196), 6)
endif
else
scroll(oBrowse:nTop, oBrowse:nLeft, oBrowse:nBottom, oBrowse:nRight)
nMidRow := oBrowse:nTop + (oBrowse:nBottom - oBrowse:nTop) / 2
nMidCol := oBrowse:nLeft + (oBrowse:nRight - oBrowse:nLeft) / 2
@ nMidRow, nMidCol - 17 say "Hey! No records for this parent!!"
lRetVal := .f.
endif

```



```

setpos(nOldRow, nOldCol)
dispend()
return IRetval

/*
  Function: Gilligan()
  Purpose: Skip function to restrict data subset for child browse
*/
static function gilligan(nSkipCnt, altems)
local nMovement := 0
do case
  // no movement
  case nSkipCnt == 0
    (altems[_ALIAS])->(dbskip(0))
  // moving forward
  case nSkipCnt > 0
    do while nMovement < nSkipCnt .and. ;
      eval(altems[_INDEXBLOCK]) <= endval ;
      .and. ! (altems[_ALIAS])->(eof())
      (altems[_ALIAS])->(dbskip())
      nMovement++
    enddo
    // make sure that we are within range -- if not, back up
    do while (eval(altems[_INDEXBLOCK]) > endval ;
      .or. (altems[_ALIAS])->(eof()) ;
      .and. ! (altems[_ALIAS])->(bof())
      (altems[_ALIAS])->(dbskip(-1))
      nMovement--
    enddo
    if (altems[_ALIAS])->(bof()) // no data in range... fall out
      keyboard chr(K_ESC)
    endif
  // moving backward
  case nSkipCnt < 0
    do while nMovement > nSkipCnt .and. ;
      eval(altems[_INDEXBLOCK]) >= startval
      (altems[_ALIAS])->(dbskip(-1))
      if (altems[_ALIAS])->( bof() )
        exit
      endif
      nMovement--
    enddo
    // make sure that we are within range -- if not, go forward
    do while eval(altems[_INDEXBLOCK]) < startval .and. ;
      ! (altems[_ALIAS])->( eof() )
      (altems[_ALIAS])->(dbskip())
      nMovement++

```

```

        enddo
        if (altems[_ALIAS])->( eof() ) // no data within range... fall out
            keyboard chr(K_ESC)
        endif
    endcase
    return nMovement

/*
    Function: EditCell()
*/
static function editcell(oBrowse)
local oColumn := oBrowse:getColumn(oBrowse:colPos)
local v := eval(oColumn:block)
local lRetval := .f.
local nOldCursor := setcursor(SC_NORMAL)
local lOldScore := set(_SET_SCOREBOARD, .f.)
readmodal( { getnew(Row(), Col(), ;
    { |_1 | if(pcount() == 0, v, v := _1) }, ;
    oColumn:heading, '@K', oBrowse:colorSpec) } )
setcursor(nOldCursor)
if v <> eval(oColumn:block) // i.e., variable was changed
    if (oBrowse:cargo[_ALIAS])->( rlock())
        eval(oColumn:block, v) // this changes the field
        (oBrowse:cargo[_ALIAS])->( dbunlock() )
        lRetval := .t.
    endif
endif
set(_SET_SCOREBOARD, lOldScore)
return lRetval

```

جدول الاستعراضات الرئيسة/الفرعية/التابعة للفرعية

استطعنا في المثال السابق مزمنة نوافذ الاستعراض الرئيسة والفرعية بحيث يتم تحديث النافذة الفرعية أثناء الانتقال في النافذة الرئيسة. وسنضيف هنا إمكانية تحديث نافذة تابعة للفرعية أثناء استعراض النافذة الرئيسة. بينما نتقل في نافذة الاسماء ستعرض النافذة الفرعية كافة الفواتير الخاصة بكل شخص (أو رسالة تفيد بعدم وجود فواتير خاصة به) وستعرض النافذة التابعة للفرعية كافة الأصناف الرئيسة المتوافقة مع الفاتورة الأولى في النافذة الفرعية. ويمكن استخدام مفتاح [Tab] للانتقال بين هذه النوافذ ،

وستكون النافذة الفعالة واضحة بسبب إطارها السميك. وإذا كانت إحدى النوافذ الفرعية لا تحتوي أية بيانات يمكن الانتقال إليها. كما يمكن استخدام القاعدة المنطقية ذاتها ، لكننا سنحتاج إلى إجراء عدة تغييرات فيها. أول تغيير هو في المتغيرين "قيمة البداية" STARTVAL و "قيمة النهاية" ENDVAL اللذين كانا ثابتين على عرض الملف. ولأنه سيكون لدينا مجموعتان جزئيتان مختلفتان من البيانات ، إحداهما للنافذة الفرعية ، والأخرى للنافذة التابعة للفرعية ، وسيخزن كل منهما في الشحنة cargo الخاصة بكل استعراض. وبما أننا لانعرف في هذه اللحظة قيمة كل من هاتين المجموعتين ، سنجعل كل "شحنة استعراض" browse: cargo مصفوفة ونترك العنصر الأول فارغاً لإدخال القيمة لاحقاً.

ولتوضيح ذلك ، سنورد فيما يلي كتلة الشيفرة التي ستلحق بالاستعراض الرئيس:

```
{ ( ( (fieldget(1) )->(parent) , browses_[2] ) showchild | ) }
```

وهذا سيمرر قيمة حقل في قاعدة بيانات الاستعراض الرئيسة إلى وظيفة "عرض السجلات الفرعية" (ShowChild) ، ثم نعين المتغير الثاني في وظيفة "عرض السجلات الفرعية" (ShowChild) إلى b: cargo[1] . وفي حال تشغيل هذه النافذة والانتقال ضمنها ، يجب الإبقاء على هذه القيمة.

ذكرنا آنفاً أنه سيكون لكل هدف من أهداف الاستعراض الرئيسة والفرعية والتابعة للفرعية مصفوفة شحنة يكون تركيبها كما يلي:

b: cargo[1] : فراغ لقيمة ندخلها لاحقاً لتحديد المجموعة الجزئية (لايستخدم من قبل النافذة الرئيسة) (ثابت البيان : SUBSET_VALUE).

b: cargo[2] : يتم تقييم كتلة الشيفرة لعرض جدول الاستعراض المرتبط (لايستخدم من قبل النافذة التابعة للفرعية) (ثابت البيان : NEXT_BROWSE).

b:cargo[3] : كتلة شيفرة الإشارة في دليل الفهرس (لا يستخدم من قبل النافذة الرئيسة). تستخدم في النوافذ الفرعية والتابعة للفرعية للتحقق من وجود سجلات صحيحة للمجموعة الجزئية المطلوبة (ثابت البيان : INDEXKEY_REF).

لاحظ أن الاستعراضات الفرعية والتابعة للفرعية تمرر كمتغيرات في وظيفة "تجاوز الكتلة" skipblock الخاصة بها. وهذا ضروري لأنه يجب أن يكون لهذه الوظيفة إمكانية الوصول إلى : (أ) قيمة البيانات التي تتحكم بالمجموعة الجزئية ، (ب) كتلة شيفرة الإشارة في دليل الفهرس ، وقد خزنت الأولى (أ) في العنصر الأول ، والثانية (ب) في العنصر الثالث من مصفوفة شحنة الاستعراض browse:cargo . وهذا يوضح عمل النموذج المصمم حسب الهدف ، فقد ضمنت كافة البيانات اللازمة في كل هدف من أهداف الاستعراض ويمكن الوصول إليها كلما كان الاستعراض مرئياً. ذكرنا أعلاه أنه عند عدم وجود سجلات فرعية متوافقة مع الرئيسة ، ستظهر رسالة بهذا الشأن. ومع ذلك سيكون هناك بيانات في النافذة التابعة للفرعية ، ولا ينبغي ذلك. والحل هو كتابة كتلة الشيفرة (cargo[2]) للاستعراض الفرعي بحيث يمكنه قبول متغير اختياري ، ثم يمرر هذا المتغير مباشرة إلى وظيفة "عرض السجلات الفرعية" ShowChild()

```
{ | 1Nodata | showchild(browses_[B_GRANDCHILD] , ;  
  (child)->(fieldget(1)) , 1Nodata) }
```

عند وصول هذا المتغير إلى الوظيفة "عرض السجلات الفرعية" ShowChild() فلن يبحث عن بيانات صحيحة ، بل سيفترض عدم وجودها. وإذا كنت تعرض جدول الاستعراض الفرعي ، فستقوم وظيفة "عرض السجلات الفرعية" ShowChild() بتقييم كتلة الشيفرة المخزنة بالاستعراض الفرعي لعرض الاستعراض التابع للفرعي. وطبعاً ستستدعي كتلة الشيفرة هذه وظيفة "عرض السجلات الفرعية" ShowChild() ثانية ، ويمكن التوسع لإيجاد مستويات للاستعراضات.

تتطلب وظيفة "الاستعراض الرئيس/الفرعي" (ParentChild) ثلاث متغيرات: <cParent> و <cChild> و <cGrandchild>. ستفتح هذه الملفات الثلاثة وتفترض أن الفهارس التي ستفتح لها أسماء قواعد البيانات ذاتها. وإذا لم تكن كذلك، غير الشيفرة بحيث تقبل أسماء الملفات في الفهارس كمتغيرات إضافية.

تحذير

يفترض هذا المثال أن الحقل الأول في قاعدتي البيانات الرئيسة والفرعية يقوم بدور حقل الدليل. ولهذا قمنا بتمرير (fieldget(1))-(parent) أو (fieldget(1))-(child) إلى وظيفة "عرض السجلات الفرعية" (ShowChild) (انظر الأسطر التي تؤسس الشحنة للاستعراضات browses_[1] و browses_[2]). قد يتطلب ذلك تغييرها للإشارة مباشرة إلى الحقل الملائم.

```
// filename : TBROW40.PRG
#include "box.ch"
#include "inkey.ch"

#define B_THICK chr(219)+chr(223)+chr(219)+chr(219)+;
chr(219)+chr(220)+chr(219)+chr(219)

#define TEST // to include the following stub program

#ifdef TEST

function tbrow40
local x
local y
local z
local testdata_ := { "Greg Lief", "Jennifer Lief", "Mary Gries", ;
"Mark Worthen", "Carol Mills", "Pat Williams", ;
"Doug Rist", "Justin Lief", "Nada" }
? "creating test data"

/*
PEOPLE.DBF serves as the parent database.
INV_TEST.DBF will be the child database.
LINETEST.DBF will be the grandchild database.
*/
```



```

dbcreate('people', { { "NAME", "C", 20, 0 } })
use people new
z := len(testdata_)
for x := 1 to z
    append blank
    people->name := testdata_[x]
next
dbcreate('inv_test', { { "INVNO", "C", 4, 0 }, ;
                      { "NAME", "C", 20, 0 } })
use inv_test new
index on inv_test->name to inv_test
for x := 1 to 79
    append blank
    people->(dbgoto(x % 8 + 1))
    inv_test->invno := str(x * 105 + 1000, 4)
    inv_test->name := people->name
next
dbcreate('linetest', { { "INVNO", "C", 4, 0 }, ;
                      { "LINENO", "N", 3, 0 }, ;
                      { "BALANCE", "N", 7, 2 } })
use linetest new
index on linetest->invno + str(linetest->lineno) to linetest
for x := 1 to 79
    inv_test->(dbgoto(x))
    for y := 1 to 4
        append blank
        linetest->invno := inv_test->invno
        linetest->lineno := y
        linetest->balance := x * y
    next
next
close data
setcursor(0)
setcolor('+w/b,n/bg')
parentchild('people','inv_test','linetest')
ferase('people.dbf')
ferase('inv_test.dbf')
ferase('inv_test.ntx')
ferase('linetest.dbf')
ferase('linetest.ntx')
return nil

#endif // TEST

#define B_PARENT      1
#define B_CHILD       2
#define B_GRANDCHILD  3

```

```
#define SUBSET_VALUE 1
#define NEXT_BROWSE 2
#define INDEXKEY_REF 3

static function parentchild(parent, child, grandchild)
local x
local z
local c
local cumbrow
local browses_
local cfield
browses_ := { TBrowseDB(1, 1, 7, maxcol()-1), ;
               TBrowseDB(10, 1, 15, maxcol()-1), ;
               TBrowseDB(18, 1, maxrow()-1, maxcol()-1) }
use (grandchild) index (grandchild) new
use (child) index (child) new
use (parent) new

//
// note: assuming that parent key field is field #1 in the DBF structure
//
cfield := (parent)->(field(1))
browses_[B_PARENT]:addColumn(tbcolumnew(cfield, fieldwblock(cfield, ;
                                select(parent))))
browses_[B_PARENT]:cargo := { NIL, { || showchild(browses_[B_CHILD], ;
                                (parent)->(fieldget(1))) } }

//
// initialize movement blocks for child browse
//
browses_[B_CHILD]:goTopBlock := { || (child)->( ;
                                dbseek(browses_[B_CHILD]:cargo[SUBSET_VALUE], .t.)) }
browses_[B_CHILD]:goBottomBlock := { || (child)->( ;
                                dbseek(substr(browses_[B_CHILD]:cargo[SUBSET_VALUE], 1, ;
                                len(browses_[B_CHILD]:cargo[SUBSET_VALUE]) - 1) + ;
                                chr( asc(right(browses_[B_CHILD]:cargo[SUBSET_VALUE], 1)) + 1 ), .t.)), ;
                                (child)->(dbskip(-1))) }
browses_[B_CHILD]:skipBlock := { | SkipCnt | gilligan(child, SkipCnt, ;
                                browses_[B_CHILD]) }

//
// using all fields from the child database... change this if you want
//
z := (child)->(fcount())
for x := 1 to z
    cfield := (child)->(field(x))
    browses_[B_CHILD]:addColumn(TBColumnNew(cfield, fieldwblock(cfield, ;
```

```

        select(child))))
next
browses_[B_CHILD]:cargo := { NIL, ;
    { | INodata | showchild(browses_[B_GRANDCHILD], ;
        (child)->(fieldget(1)), INodata) },;
    &("{ || " + child + "->(" + (child)->(indexkey(0)) + ") }") }

//
// initialize movement blocks for grandchild browse
//
browses_[B_GRANDCHILD]:goTopBlock := { || (grandchild)->( ;
    dbseek(browses_[B_GRANDCHILD]:cargo[SUBSET_VALUE], .t.)) }
browses_[B_GRANDCHILD]:goBottomBlock := { || (child)->( ;
    dbseek(substr(browses_[B_GRANDCHILD]:cargo[SUBSET_VALUE], 1, ;
        len(browses_[B_GRANDCHILD]:cargo[SUBSET_VALUE]) - 1) + ;
        chr(asc(right(browses_[B_GRANDCHILD]:cargo[SUBSET_VALUE], 1)) + 1),
    .t.)),;
    (child)->(dbskip(-1)) }
browses_[B_GRANDCHILD]:skipBlock := { | SkipCnt | ;
    gilligan(grandchild, SkipCnt, browses_[B_GRANDCHILD]) }
browses_[B_GRANDCHILD]:cargo := { NIL, , ;
    &("{ || " + grandchild + "->(" + (grandchild)->(indexkey(0)) + ") }") }

//
// using all fields from grandchild database... change this if you want
//
z := (grandchild)->(fcount())
for x := 1 to z
    cfield := (grandchild)->(field(x))
    browses_[B_GRANDCHILD]:addColumn(TBColumnNew(cfield,
        fieldwblock(cfield, ;
            select(grandchild))))
next

currrow := 1
dispbegin()
scroll()
for x := 1 to 3
    @ browses_[x]:nTop - 1, browses_[x]:nLeft - 1, ;
    browses_[x]:nBottom + 1, browses_[x]:nRight + 1 box B_SINGLE
next
dispend()
do while lastkey() <> K_ESC
    @ browses_[currrow]:nTop - 1, browses_[currrow]:nLeft - 1, ;
    browses_[currrow]:nBottom + 1, browses_[currrow]:nRight + 1 ;
    box B_THICK
    dobrowse(browses_, @currrow)

```

```
enddo  
return nil
```

```
/*  
Function: DoBrowse( <aBrowses>, <nPointer> )  
Purpose: Browse display/keystroke workhorse  
Parameters: <aBrowses> -- array containing all three browses  
            <nPointer> -- # of current browse in <aBrowses>  
Returns: Nada  
*/  
function dobrowse(browses_, pointer)  
local key  
local b := browses_[pointer]  
local lcanswap  
do while key <> K_ESC .and. key <> K_TAB  
do while (key := inkey()) == 0 .and. ! b:stabilize()  
enddo  
lcanswap := .t.  
if key == 0  
if ! empty(b:cargo) .and. valtype(b:cargo[NEXT_BROWSE]) == "B"  
lcanswap := eval(b:cargo[NEXT_BROWSE])  
endif  
key := inkey(0)  
endif  
do case  
case key == K_UP  
b:up()  
case key == K_DOWN  
b:down()  
case key == K_PGUP  
b:pageUp()  
case key == K_PGDN  
b:pageDown()  
case key == K_LEFT  
b:left()  
case key == K_RIGHT  
b:right()  
case key == K_TAB .and. lcanswap  
@ b:nTop - 1, b:nLeft - 1, ;  
b:nBottom + 1, b:nRight + 1 box B_SINGLE  
if pointer == 3  
pointer := 1  
else  
pointer++  
endif  
case key == K_ENTER
```



```

//
// add code to edit current cell if you want
//
endcase
enddo
return nil

/*
Function: ShowChild( <browse>, <value>, <INodata> )
Purpose:  Display child or grandchild browse without key processing
          Synchronizes the child window to update in
          accordance with changes in the parent window
Parameters: <browse> -- child/grandchild TBrowse object
            <value> -- data to base the subset upon
            <INodata> -- logical override... if there are no
            child records matching the parent, this flag will force
            the grandchild window to be shown as empty
Returns:  Logical -- .T. if there are valid records in the subset
          This enables the TAB key to switch to this window (see
          the logic above in DoBrowse())
*/
function showchild(b, val, INodata)
local keyval
local r,c
local midrow, midcol
local ret_val := .t.
if INodata == NIL
    INodata := .f.
endif
if ! INodata
    b:cargo[SUBSET_VALUE] := val
    dispbegin()
    b:goTop()
    keyval := eval(b:cargo[INDEXKEY_REF])
endif
/*
    Make sure there are actually valid child records...
    NOTE: we cannot use the exact equality (==) operator because then the
    comparison will fail if we are using concatenated keys for the child
    database (and as fate would have it, that will usually be the case).
*/
if ! INodata .and. keyval = val
    b:refreshAll()
    do while ! b:stabilize()
    enddo
else

```



```

scroll(b:nTop, b:nLeft, b:nBottom, b:nRight)
r := row()
c := col()
midrow := b:nTop + (b:nBottom - b:nTop) / 2
midcol := b:nLeft + (b:nRight - b:nLeft) / 2
@ midrow, midcol - 8 say "No records found"
setpos(r, c)
ret_val := .f.
endif
if ! empty(b:cargo) .and. valtype(b:cargo[NEXT_BROWSE]) == "B"
    //----- pass return value to code block because we might need it to
    //----- force the grandchild window not to show any records (in the
    //----- event that there are no child records for the current parent)
    eval(b:cargo[NEXT_BROWSE], ! ret_val)
endif
dispend()
return ret_val

/*
Function:  Gilligan( <cAlias>, <nSkipcnt>, <oBrowse> )
Purpose:   Custom skip function for child and grandchild browses
Parameters: <cAlias> -- alias in which to skip around
            <nSkipcnt> -- rows TBrowse thinks it has to move (aha,
                        but we're going to change that, aren't we?)
            <oBrowse> -- corresponding browse object -- necessary so
                        we can have access to the first and third
                        elements of the cargo array (data
                        controlling the subset, and indexkey
                        reference block, respectively)
Returns:   Number of rows for TBrowse to move
*/
static function gilligan(alias, skipcnt, b)
local movement := 0
local bVal := b:cargo[INDEXKEY_REF]
do case
    case skipcnt == 0
        (alias)->(dbskip(0))
    case skipcnt > 0
        do while movement < skipcnt .and. eval(bVal) <= b:cargo[SUBSET_VALUE] ;
            .and. ! (alias)->(eof())
            (alias)->(dbskip())
            movement++
        enddo
        // make sure that we are within range -- if not, move backward

```

```

do while (eval(bVal) > b:cargo[SUBSET_VALUE] .or. (alias)->(eof())) ;
    .and. ! (alias)->(bof())
    (alias)->(dbskip(-1))
    movement--
enddo
if (alias)->(bof())      // no data in range... fall out
    keyboard chr(K_ESC)
endif
case skipcnt < 0
    do while movement > skipcnt .and. eval(bVal) >= b:cargo[SUBSET_VALUE]
        (alias)->(dbskip(-1))
        if (alias)->( bof() )
            exit
        endif
        movement--
    enddo
    // make sure that we are within range -- if not, move forward
    do while eval(bVal) < b:cargo[SUBSET_VALUE] .and. ! (alias)->( eof() )
        (alias)->(dbskip())
        movement++
    enddo
    if (alias)->( eof() )      // no data within range... fall out
        keyboard chr(K_ESC)
    endif
endcase
return movement

```

الانتقال من مجموعة جزئية إلى أخرى

السؤال التالي طرحه علينا أحد المبرمجين العام الماضي: "إذا كان لديك فهرس تحكم وكنت تعمل على قاعدة بيانات كبيرة ، فهل هنالك من طريقة لجعل جدول الاستعراض TBrowse يقفز سريعاً إلى مفتاح الفهرس التالي أو السابق؟". بكل تأكيد، إذ يستلزم القيام بذلك استخدام "كتلة شيفرة الإشارة في دليل الفهرس" التي وردت في المثال الأخير. سنقوم ثالية بتخزين كتلة الشيفرة هذه في المتغير الفوري "شحنة الاستعراض" TBrowse:cargo

الخطوة التالية هي تجهيز وظيفة "القفز إلى الأعلى" (JumpUp) ووظيفة "القفز إلى أسفل" (JumpDown) التي سترجع إلى كتلة شيفرة الفهرس هذه وتقوم بتقييمها حسب الطلب. تستخدم وظيفة "القفز إلى الأعلى" (JumpUp) للانتقال إلى بداية المجموعة الجزئية السابقة من البيانات ، بينما تنتقل وظيفة "القفز إلى أسفل" (JumpDown) إلى بداية المجموعة الجزئية التالية. تمرر هدف استعراض TBrowse لكل من هذين الوظيفتين فيستطيعان الرجوع إلى المتغير الفوري "الشحنة" cargo الخاصة بهذا الهدف.

تحتفظ وظيفة "القفز إلى أسفل" (JumpDown) بالموضع الحالي لمؤشر السجل ، ثم تقوم بتقييم كتلة شيفرة الإشارة في دليل الفهرس وتخزن القيمة. وبعدها تنفذ "SOFTSEEK" للموضع المنطقي في قاعدة البيانات. لتحديد الموضع المنطقي التالي بالبيانات الحرفية يجب أن نزيد الرمز في أقصى اليمين من مجموعة الرموز بقيمة واحدة من شيفرة آسكي ASCII مثل مايلي:

```
substr(value, 1, len(value) - 1) + chr(asc(right(value, 1)) + 1)
```

تمرر هذه القيمة ، إضافة إلى القيمة المنطقية "حقيقي" (T.) إلى وظيفة "البحث في قاعدة البيانات" (DBSEEK). ويوجه المتغير الثاني هذا الوظيفة لتنفيذ (SOFTSEEK) ، بحيث نتجنب الانتقال إلى نهاية الملف وإخفاق البرنامج في حال عدم وجود سجل مطابق. إذا لم يكن الموضع الحالي عند نهاية الملف ، فيمكن تجديد نافذة الاستعراض بوظيفة "تجديد كافة البيانات" (refreshAll) ، وإلا فنعيد مؤشر السجل إلى موضعه السابق.

تقوم وظيفة "القفز إلى الأعلى" (JumpUp) ، بتقييم كتلة شيفرة الإشارة في دليل الفهرس وتبحث عن المجموعة الجزئية المناسبة وتنتقل إلى بدايتها. ثم تتجاوز سجلاً واحداً باتجاه الخلف (أي باتجاه بداية الملف) للانتقال إلى المجموعة الجزئية السابقة من البيانات.

ثم يتم تقييم كتلة شيفرة الإشارة في دليل الفهرس ثانية استعداداً "للبحث" عن بداية هذه المجموعة الجزئية السابقة. ولكن هناك خطأ معروفاً في برنامج الاستعراض TBrowse يحدث عند البحث للانتقال إلى صف أعلى في نافذة الاستعراض ذاتها. وللتغلب على هذه المشكلة قبل إجراء عملية البحث الثانية نستخدم وظيفة "الانتقال إلى أعلى" في الاستعراض (TBrowsegoTop) للانتقال إلى أعلى الملف. أخيراً تجدد وظيفة "القفز إلى الأعلى" (JumpUp) نافذة الاستعراض بواسطة وظيفة "تجديد كافة البيانات" (refreshAll).

يبين المثال التالي وظيفة "القفز إلى الأعلى" (JumpUp) ووظيفة "القفز إلى أسفل" (JumpDown) اللذين يمكن استخدامهما بواسطة مفتاحي [Home] و [End] على التوالي. لاحظ المتغيرين الاختياريين "قيمة البداية" <startval> و "قيمة النهاية" <endval> اللذين يتيحان لك قصر الاستعراض على مجموعة جزئية محددة من البيانات. فإذا حددت هذين المتغيرين ستُنشأ كتل انتقال خاصة في الاستعراض لقصر الوصول إلى تلك المجموعة الجزئية. لاحظ أيضاً إضافة قاعدة منطقية للسلامة في وظيفة "القفز إلى الأعلى" (JumpUp) ووظيفة "القفز إلى أسفل" (JumpDown) لضمان عدم القفز وراء حدود المجموعة الجزئية بغير قصد.

```
// filename : TBROW41.PRG
#include "box.ch"
#include "inkey.ch"

#xtranslate SCRNCENTER(<row>, <msg> [, <color> ] ) => ;
    SetPos( <row>, int((( maxcol()+1 - len(<msg>)) / 2)) ) ;
    DispOut( <msg> [, <color>] )

function tbrow41(cDbf, cNtx, startval, endval)
local x
local oBrowse
local searchval
local nFields
local nKey
local cField
local cOldColor
local nOldCursor
if cDbf == NIL .or. cNtx == NIL
```



```
? "Syntax: JUMPDEMO <dbfname> <nixname> [<start> <end>]"
? "For the purposes of this demo, the indexkey should be character."
? "Optional parameters <start> and <end> restrict the data subset."
return .f.
endif
cOldColor := setcolor("n/bg, +w/r")
nOldCursor := setcursor(0)
oBrowse := TBrowseDB(1, 1, maxrow()-1, maxcol()-1)
use (cDbf) index (cNtx)

//----- create indexkey reference codeblock and store in b:cargo i-var
oBrowse:cargo := &("{ || " + indexkey(0) + "}")
if valtype(eval(oBrowse:cargo)) <> "C"
? "This example is written specifically for character data..."
? "Please try it again with an index file containing a character index key"
use
return .f.
endif

//----- create columns for each field in database
nFields := fcount()
for x := 1 to nFields
cField := field(x)
oBrowse:AddColumn(TBColumnNew(cField, fieldblock(cField)))
next

//----- optional cosmetic niceties
oBrowse:colSep := chr(32) + chr(179) + chr(32)
oBrowse:headSep := chr(205) + chr(209) + chr(205)
scroll()
@ oBrowse:nTop - 1, oBrowse:nLeft - 1, ;
oBrowse:nBottom + 1, oBrowse:nRight + 1 box B_SINGLE + ''

//----- tweak custom movement blocks if subset parameters were specified
if startval <> NIL .and. endval <> NIL
searchval := substr(endval, 1, len(endval) - 1) + ;
chr( asc(right(endval, 1)) + 1 )
oBrowse:goTopBlock := { || dbseek(startval, .t.) }
oBrowse:goBottomBlock := { || dbseek(searchval, .t.), dbskip(-1) }
oBrowse:skipBlock := { | nSkipCnt | gilligan(nSkipCnt, ;
oBrowse:cargo, startval, endval) }
eval(oBrowse:goTopBlock)
ScrnCenter(oBrowse:nTop - 1, "[ Viewing data between " + startval + ;
" and " + endval + " ]", "+w/bg")
endif
```



```
ScrnCenter(oBrowse:nBottom + 1, "[ Press HOME and END to jump to start " + ;  
          "of previous/next subgroup ]", "+w/bg")
```

```
//----- da main loop...
```

```
do while nKey <> K_ESC  
  dispbegin()  
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()  
    enddo  
  dispend()  
  if oBrowse:stable  
    nKey := inkey(0)  
  endif  
  do case  
    case nKey == K_UP  
      oBrowse:up()  
      oBrowse:stabilize()  
      if oBrowse:hitTop  
        alert("Top of data")  
      endif  
    case nKey == K_DOWN  
      oBrowse:down()  
      oBrowse:stabilize()  
      if oBrowse:hitBottom  
        alert("Bottom of data")  
      endif  
    case nKey == K_HOME .and. oBrowse:stable  
      JumpUp(oBrowse, startval)  
    case nKey == K_END .and. oBrowse:stable  
      JumpDown(oBrowse, endval)  
    otherwise  
      keytest(nKey, oBrowse)  
  endcase  
enddo  
use  
setcolor(cOldColor)  
setcursor(nOldCursor)  
return nil
```

```
/*  
  Gilligan(): custom skipBlock function for use w/ data subsets  
*/  
static function gilligan(skipcnt, val, startval, endval)  
  local nMovement := 0  
  do case  
    // no movement
```

```
case skipcnt == 0
  skip 0
// moving forward
case skipcnt > 0
  do while nMovement < skipcnt .and. eval(val) <= endval
    skip 1
    nMovement++
  enddo
// make sure that we are within range -- if not, move backward
do while (eval(val) > endval .or. eof()) .and. ! bof()
  skip -1
  nMovement--
enddo
if bof()      // no data in range...
  keyboard chr(K_ESC)
endif
// moving backward
case skipcnt < 0
  do while nMovement > skipcnt .and. eval(val) >= startval
    skip -1
    if bof()
      exit
    endif
    nMovement--
  enddo
// make sure that we are within range -- if not, move forward
do while eval(val) < startval .and. ! eof()
  skip
  nMovement++
enddo
if eof()      // no data within range...
  keyboard chr(K_ESC)
endif
endcase
return nMovement

//----- end static function Gilligan()

//-----//

/*
  JumpDown(): jump to beginning of next subgroup (active index only!)
*/
static function jumpdown(oBrowse, endval)
local val := eval(oBrowse:cargo)
local nRec := recno()
```

```
//----- jump only if we are not passing the end of the data subset
if endval == NIL .or. endval >= val
  dbseek( substr(val, 1, len(val) - 1) + ;
    chr( asc(right(val, 1)) + 1 ), .t.)
  if eof()
    go nRec
  else
    oBrowse:refreshAll()
  endif
endif
return nil

//----- end static function JumpDown()

//-----//

/*
  JumpUp(): jump to beginning of previous subgroup (active index only!)
*/
static function jumpup(oBrowse, startval)
local val
//----- jump only if we are not passing the start of the data subset
if startval == NIL .or. eval(oBrowse:cargo) > startval
  dbseek(eval(oBrowse:cargo), .t.)
  dbskip(-1)
  val := eval(oBrowse:cargo)
  oBrowse:goTop()
  dbseek(val, .t.)
  oBrowse:refreshAll()
endif
return nil
```

أهداف الاستعراض الساكنة Static TBrowse Objects

نستخدم لتدقيق البيانات جداول البحث look-up tables . وفي كليبر Summer'87
 كنا نستخدم عادة () DBEDIT كأساس في عملية البحث والتدقيق. أما في كليبر
 5.x فإننا تحولنا لاستخدام جدول الاستعراض TBrowse.

الطريقة العامة لذلك هي إعادة إنشاء هدف جدول الاستعراض TBrowse كلما دخلنا إلى وظيفة البحث والتدقيق:

```
function lookup(....)
local browse := tbrowsedb(...)
local column := tbcolumnnew(...)
browse:addColumn(column)
// stabilize
// deal with keypress
// paste selected value from lookup table into the GET
return nil
```

وتجنباً لتكرار هذا الإجراء في كل مرة ، يمكننا جعل هدف الاستعراض "ساكنة" STATIC بدلاً من "متغير محلي" LOCAL ، وبذلك يحتفظ الهدف بقيمته طيلة مدة تشغيل البرنامج. وفيما يلي الجزء الأول في القاعدة المنطقية:

```
function lookup
static browse
....
if browse == NIL
    browse := TBrowseDB(8, 1, maxrow() - 1, )
    browse:headSep := chr(196)
else
    browse:delColumn(1)
endif
```

إن كل مايقع ضمن مدى "ساكنة" STATIC يؤسس بقيمة "الصفري" NIL. وبذلك يمكننا بسرعة معرفة إذا استخدمنا هذه الوظيفة قبل ذلك أم لا ، لأن الاستعراض لم يعد صفراً NIL ، وإن كان صفراً ننشئ هدف الاستعراض.

إذا كان هدف الاستعراض موجوداً سابقاً نحذف العمود الذي ضمنه. ذلك لأن وظيفة "البحث والتدقيق" (look-up) شاملة ويمكنها الإشارة إلى أنواع مختلفة من الحقول. وإذا استدعينا هذا الوظيفة ثانية لحقل مختلف فلن ينفعا النظر إلى الحقل السابق.

إذا كنت تعرف مسبقاً بأنك ستستخدم وظيفة "البحث والتدقيق" (look-up) في الحقل ذاته فلا تحذف العمود في كل مرة ، لأن الأداء سيكون أفضل بدون إعادة إنشاء هدف وظيفة عمود الاستعراض (TBColumn).

لاحظ أنه يجب علينا تعديل عدة متغيرات فورية في الاستعراض كلما دخلنا هذه الوظيفة. فيجب تغيير المتغير browse:nRight لأن عرض العمود سيتغير ، ويجب إعادة ضبط المتغير "الموضع في الصف" browse:rowPos إلى (١) في كل مرة بحيث لا يبدأ المستخدم من منتصف نافذة البحث والتدقيق.

وأخيراً يجب استخدام وظيفة "تجديد كافة البيانات" (refreshAll) حتى يعاد عرض البيانات بشكل صحيح في نافذة الاستعراض TBrowse. ولمعرفة مدى أهمية ذلك ضع خطين مائلين (//) في السطر لإيقاف تشغيله.

مع أن المثال بسيط إلا أنه سيبين مدى أهمية هذا المفهوم الذي سيحسن الأداء. وسنلاحظ أننا نحتاج لإعادة إنشاء أهداف الاستعراض في العديد من الوظائف في كل مرة ندخل وظيفة "البحث والتدقيق" (look-up).

```
// filename : TBROW44.PRG
#include "inkey.ch"
```

```
function tbrow44
local x
local nOldcursor := setcursor(0)
local cOldcolor := setcolor("+w/b")
```

```
//----- dummy data for purpose of this example
local aData := { { "Greg Lief", "Grumpfish, Inc.", "Salem, OR" }, ;
  { "Joe Booth", "CLIPWKS", "Philadelphia, PA" }, ;
  { "Mike Mussina", "Baltimore Orioles", "" }, ;
  { "Mary Gries", "Grumpfish, Inc.", "Salem, OR" }, ;
  { "Ira Emus", "Extrasensory Software", "Los Angeles, CA" }, ;
  { "Tom Glavine", "Atlanta Braves", "" }, ;
  { "Bill Gates", "Microsoft Corp.", "Redmond, WA" }, ;
  { "Cal Ripken", "Baltimore Orioles", "" }, ;
  { "Dave Rifkind", "Extrasensory Software", "Los Angeles, CA" } }
```

```
//----- create temporary database
```



```

dbcreate("tbrow44", { { "NAME", "C", 26, 0 } , ;
                      { "COMPANY", "C", 26, 0 } , ;
                      { "ADDRESS", "C", 26, 0 } } )
use tbrow44 new
for x := 1 to len(aData)
  append blank
  aeval(aData[x], { | ele, count | fieldput(count, ele) } )
next
scroll()
MultLine()
setcolor(cOldcolor)
setcursor(nOldcursor)
use
ferase("tbrow44.dbf")
return nil

// end main stub program
//-----//

/*
  Function: MultLine()
*/
static function MultLine
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey := 0
local nCurrEle := 1
local nMaxEle
local lFixUp := .t.

oBrowse:skipBlock := { | nSkipCnt | Gilligan(nSkipCnt, @nCurrEle, ;
                      nMaxEle, oBrowse:getColumn(oBrowse:colPos)) }
oBrowse:colorSpec := "+W/B, +W/R, W/G, +W/G"

nMaxEle := fcount() + 1
oColumn := TBColumnNew("Info", { | | { tbrow44->name, ;
                      tbrow44->company, tbrow44->address, ;
                      replicate(chr(196), 25))[nCurrEle] } } )

//----- draw separators in different color than data
oColumn:colorBlock := { || if(nCurrEle == nMaxEle, {3, 4}, {1, 2} ) }

oBrowse:AddColumn(oColumn)

```

```
//---- second column shows record # to prove that this beast works!
oColumn := TBColumnNew("Rec #", ;
    { || if(nCurrEle <> nMaxEle, recno(), ' ') } )
oBrowse:AddColumn(oColumn)
go top
```

```
//---- main keypress loop begins...
do while nKey <> K_ESC
    do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
        enddo
    //---- if this is the first time in, we must reset element pointer
    //---- which will have been torn asunder by the skipBlock below
    if IFixUp
        nCurrEle := 1
        IFixUp := .f.
    endif
    if nKey == 0
        nKey := inkey(0)
    endif
    do case
        case nKey == K_UP
            oBrowse:up()
        case nKey == K_DOWN
            oBrowse:down()
        case nKey == K_CTRL_PGUP
            oBrowse:goTop()
            nCurrEle := 1
            IFixUp := .t.
        case nKey == K_CTRL_PGDN
            oBrowse:goBottom()
            nCurrEle := nMaxEle
        endcase
    enddo
return nil
```

```
// end static function MultLine()
//-----//
```

```
/*
    Function: Gilligan()
    Purpose: Custom skipblock function for MultLine()
*/
static function Gilligan(nSkipCnt, nEle, nMaxEle, oColumn)
local nMovement := 0
do case
    // no nMovement
```

```
case nSkipCnt == 0
  skip 0
// moving forward
case nSkipCnt > 0
  do while nMovement < nSkipCnt .and. ! eof()
    if nEle < nMaxEle
      nEle++
      //----- if this cell is empty, skip past it without displaying
      if empty( eval(oColumn:block) )
        nEle++
      endif
    else
      skip 1
      nEle := 1
    endif
    nMovement++
  enddo
  if eof()
    go bottom
    nEle := nMaxEle
    nMovement--
  endif
// moving backward
case nSkipCnt < 0
  do while nMovement > nSkipCnt .and. ! bof()
    if nEle == 1
      nEle := nMaxEle
      skip -1
    else
      nEle--
      //----- if this cell is empty, skip past it without displaying
      if empty( eval(oColumn:block) )
        nEle--
      endif
    endif
    nMovement--
  enddo
  if bof()
    go top
    nEle := 1
    nMovement++
  endif
endcase
return nMovement

// end static function Gilligan()
```

تحسين الانزلاق العمودي في جدول TBrowse

إذا ضبطنا جدول الاستعراض TBrowse بعدة أعمدة (٥٠ فأكثر ، مثلاً) سنلاحظ أن التحرك الأفقي اسرع بكثير من التحرك العمودي. سبب هذا هو أن TBrowse يحفظ محتويات كافة الصفوف المرئية وكافة الأعمدة (المرئية أو المخفية) في سلسلة حرفية واحدة في الذاكرة المؤقتة . وتخزن هذه السلسلة الكبيرة داخلياً ولا يمكن الوصول إليها كواحد من المتغيرات الفورية التي تحتويها. (هذا يفسر أيضاً رسالة الخطأ "تجاوز الحد Limit Exceeded" الذي مازال غامضاً. السبب ببساطة هو أن السلسلة الحرفية هذه قد تجاوزت حد طول السلسلة في كليبر وهو (64k)).

وبما أن السلسلة الحرفية هذه تشتمل على محتويات جميع الأعمدة ، بما فيها المخفية ، فيمكن أن يكون التحرك الأفقي سريعاً جداً لأنه يمكن الوصول إلى كافة المعلومات دون تقييم كافة كتل الأعمدة. وكلما كان عدد الأعمدة في هدف الاستعراض أكبر كانت هذه السلسلة أكبر. إن لحجم السلسلة تأثير مباشر على سرعة التحرك العمودي ضمن نافذة الاستعراض. وكلما أضفنا صفوفاً مرئية جديدة يجب إعادة إنشاء المجموعة ، مما يعني البحث في جميع الأعمدة.

ولرفع أداء التحرك العمودي في جدول الاستعراض TBrowse إلى المستوى الأفضل وضعنا الفكرة التالية وهي: "أشياء فقط الأعمدة التي ستكون مرئية على الشاشة" ، ثم انتقل يساراً أو يميناً لجعل أعمدة أخرى مرئية وأضف واحذف أعمدة حسب الحاجة.

سننشئ في المثال التالي قاعدة بيانات اختبار (BIGSTRU.DBF) تحتوي (٢٠٠) حقلاً. حيث أننا نريد استعراض كافة الحقول ، لذا سنحتاج إلى (٢٠٠) عمود. ولكن بدلاً من تحميل هدف جدول الاستعراض بـ: (٢٠٠) عمود سنحمل

مصفوفة "الأعمدة" `aColumns` بـ: (٢٠٠) هدف عمود ، ثم نشير إلى هذا المصفوفة عندما نريد تحميل أعمدة في هدف جدول الاستعراض `TBrowse`.

سنستخدم المتغيرات الساكنة `static` على مدى الملف وعددها ستة :

المتغير	التوضيح
<code>aColumns</code>	موضع المؤشر في مصفوفة الأعمدة
<code>nleftCol</code>	متغير العمود الأيسر
<code>nRightCol</code>	متغير العمود الأيمن
<code>nColWidth</code>	عرض العمود
<code>nBrowseWidth</code>	عرض جدول الاستعراض
<code>nColSepWidth</code>	عرض فواصل الأعمدة

تبدأ وظيفة "تحميل الاستعراض" (`LoadBrowse`) من العمود رقم (١) وتحميل عدداً من الأعمدة بقدر ما يمكن تحميله ضمن عرض نافذة جدول الاستعراض (`nBrowseWidth`). يزيد "عرض الأعمدة" `nColWidth` عند إضافة كل عمود. كما نأخذ بعين الاعتبار عرض فواصل الأعمدة (`nColSepWidth`). (نفترض في المثال التالي أن يكون العرض متساوياً في كافة فواصل الأعمدة ، وهذا ينطبق على معظم الحالات) لا يحدد عرض العمود بأخذ القيمة من المتغير الفوري "عرض عمود الاستعراض" `TBColumn:Width` ، ولكن باستخدام وظيفة "عرض عمود الاستعراض" (`TBrowse:colWidth`).

عندما يبلغ عرض العمود أقصى حد (أي "عرض العمود" `nColWidth` أكبر من "عرض الاستعراض" `nBrowseWidth`) نحذف آخر عمود مضاف ، ونضبط متغيري "العمود الأيسر" `nLeftCol` و "العمود الأيمن" `nRightCol` بالقيم المناسبة بحيث نعرف موضع المؤشر في مصفوفة الأعمدة `aColumns`.

وسنستخدم قاعدة منطقية إضافية للمفاتيح التي ستجعل الأعمدة الجديدة مرئية:

■ **السهم الأيسر:** إذا لم يكن العمود الحالي آخر عمود مرئي من جهة اليسار ،
نستخدم وظيفة `b:left()` . أما إذا كنا في آخر عمود مرئي من جهة اليسار ، ولسنا
في العمود الأول في مصفوفة الأعمدة `aColumns` ، فعلى إدراج عمود جديد في
الحيز الأول في الاستعراض `TBrowse` . ويتم هذا باستخدام وظيفة "إدراج عمود
في الاستعراض" `TBrowse:insColumn()` . ثم نحذف أعمدة من الجانب الأيمن
من هدف الاستعراض لنخفض العرض التراكمي (`nColWidth`) لأقل من العرض
الإجمالي لناقل الاستعراض. لا يمكن افتراض أن إضافة عمود واحد يستلزم حذف
عمود واحد فقد يكون عرض العمود المضاف كبيراً بحيث يستلزم حذف عدة
أعمدة ضيقة.

■ **السهم الأيمن:** يشبه السهم الأيسر تماماً ما عدا أن إضافة الأعمدة تكون في الجانب
الأيمن والحذف في الجانب الأيسر.

■ **`[Ctrl]-[Home]`:** يجعل هذان المفتاحان العمود رقم (١) مرئياً. وقد لا يكون العمود
رقم (١) في هدف الاستعراض عند ضغط هذين المفتاحين ، فيجب في هذه الحالة
مسح جدول الاستعراض `TBrowse` وتحميله ثانية والبحث عن أعمدة في
(`aColumns`) ، تبدأ بالرقم (١). ويتم انجاز ذلك بواسطة الوظيفة "تحميل جدول
الاستعراض" `LoodBrowse()` .

■ **`[Ctrl]-[End]`:** يجعل هذان المفتاحان آخر عمود (أو العمود ذي أعلى رقم) مرئياً.
أيضاً قد لا يكون العمود ذو أعلى رقم في هدف الاستعراض ، فيجب في هذه الحالة
مسح الاستعراض وتحميله ثانية. ولكن يجب تحميله بترتيب معاكس لذلك
نستخدم وظيفة "تحميل جدول الاستعراض" `LoodBrow2()` . لأننا نسحب
الأعمدة بترتيب تنازلي في (`aColumns`) فإنها تحمل في هدف الاستعراض
بالترتيب ذاته ، وهذا يؤدي إلى نسخة مرآة (معاكسة) عند العرض. ومع ذلك
عندما نصل إلى الحد الأقصى للعرض نقلب إتجاه الأعمدة في هدف الاستعراض

بحيث تعرض على التوالي (أي بترتيب تصاعدي). وهذا ضروري لأن الأعمدة يجب أن تكون جزءاً من هدف الاستعراض من أجل أن تقوم وظيفة ColWidth() باستنتاج عرض كل منها.

هذه الطريقة مفيدة جداً ومع أن أداء التحرك الأفقي سيكون أبطأ بقليل إلا أننا سنكسب سرعة في التحرك العمودي وخاصة عند استعراض قواعد بيانات تشمل عدداً كبيراً من التراكيب.

```
// filename: TBROW43.PRG
#include "inkey.ch"
```

```
static aColumns      // repository for all column objects
static nLeftCol      // current leftmost column (in aColumns)
static nRightCol     // current rightmost column (in aColumns)
static nColWidth     // current cumulative column width
static nBrowseWidth  // width of browse window (b:nRight-b:nLeft+1)
static nColSepWidth  // width of column separator
```

```
function tbrow43
local nFields
local x
local y
local cRecno
local cField
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local nKey
local nOldcursor := setcursor(0)
local aFields
if ! file('bigstru.dbf')
    ? "Creating test database"
    aFields := array(200)
    for x := 1 to 200
        ?? "."
        aFields[x] := { "F" + ltrim(str(x)), "C", if(x % 2 == 0, 3, 6), 0 }
    next
    dbcreate('bigstru', aFields)
    use bigstru new
    for x := 1 to 99
        ?? ""
        append blank
        cRecno := str(recno(), 2)
        for y := 1 to 200
            fieldput(y, cRecno)
```

```

    next
  next
  go top
else
  use bigstru new
endif
nFields := fcount()
oBrowse:headSep := chr(205) + chr(209) + chr(205)
oBrowse:colSep := chr(32) + chr(179) + chr(32)
nBrowseWidth := oBrowse:nRight - oBrowse:nLeft + 1
nColSepWidth := len(oBrowse:colSep)
aColumns := array(nFields)

//----- load array with column objects for each field
for x := 1 to nFields
  cField := field(x)
  aColumns[x] := TBColumnNew(cField, fieldblock(cField))
next

LoadBrowse(oBrowse)

do while nKey <> K__ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
  enddo
  if nKey == 0
    nKey := inkey(0)
  endif
  do case
    case nKey == K__UP
      oBrowse:up()
    case nKey == K__DOWN
      oBrowse:down()
    case nKey == K__LEFT .and. oBrowse:stable
      if oBrowse:colPos <> oBrowse:leftVisible
        oBrowse:left()
      elseif oBrowse:colPos == oBrowse:leftVisible .and. nLeftCol > 1
        //----- insert new column on left side of browse, and
        //----- decrement left column pointer in aColumns array
        oBrowse:insColumn(1, aColumns[--nLeftCol])
        nColWidth += oBrowse:colWidth(1) + nColSepWidth
        x := oBrowse:colCount
        y := 1
        //----- delete columns from the right side of the browse
        //----- to get the cumulative width back in line with

```

```

//----- the total width of the browse window
do while nColWidth > nBrowseWidth
    nColWidth -= (oBrowse:colWidth(x) + nColSepWidth)
    oBrowse:delColumn(x--)
    y++
enddo
nRightCol -= -y
oBrowse:left()
endif
case nKey == K_RIGHT .and. oBrowse:stable
    if oBrowse:colPos <> oBrowse:rightVisible
        oBrowse:right()
    elseif oBrowse:colPos == oBrowse:rightVisible .and. ;
        nRightCol < nFields
        //----- insert new column on right side of browse, and
        //----- increment right column pointer in aColumns array
        oBrowse:addColumn(aColumns[++nRightCol])
        nColWidth += oBrowse:colWidth(oBrowse:colCount) + nColSepWidth
        x := 1
        //----- delete columns from the left side of the browse
        //----- to get the cumulative width back in line with
        //----- the total width of the browse window
        do while nColWidth > nBrowseWidth
            nColWidth -= (oBrowse:colWidth(x) + nColSepWidth)
            oBrowse:delColumn(x++)
        enddo
        nLeftCol += -x
        oBrowse:right()
    endif
case nKey == K_CTRL_HOME
    if nLeftCol > 1
        //----- must clear, then reload browse from column #1
        ClearBrowse(oBrowse)
        LoadBrowse(oBrowse)
        oBrowse:colPos := 1
    elseif oBrowse:colPos <> oBrowse:leftVisible
        oBrowse:panHome()
    endif
case nKey == K_CTRL_END
    if nRightCol < nFields
        //----- must clear, then reload browse from highest column
        ClearBrowse(oBrowse)

```



```
LoadBrow2(oBrowse, nFields)
  oBrowse:colPos := oBrowse:colCount
elseif oBrowse:colPos <> oBrowse:rightVisible
  oBrowse:panEnd()
endif
case nKey == K_HOME
  oBrowse:home()
case nKey == K_END
  oBrowse:end()
case nKey == K_PGUP
  oBrowse:pageUp()
case nKey == K_PGDN
  oBrowse:pageDown()
case nKey == K_CTRL_PGUP
  oBrowse:goTop()
case nKey == K_CTRL_PGDN
  oBrowse:goBottom()
endcase
enddo
use
setcursor(nOldcursor)
return nil

static function LoadBrowse(oBrowse)
local x := 1
nColWidth := 0
//----- add columns and increment total cumulative column width
do while nColWidth <= nBrowseWidth
  oBrowse:addColumn( aColumns[x++] )
  nColWidth += oBrowse:colWidth(oBrowse:colCount) + nColSepWidth
enddo
//----- set left and right column pointers (for aColumns array)
nLeftCol := 1
nRightCol := oBrowse:colCount - 1
//----- lop off rightmost column in browse object
nColWidth -= (oBrowse:colWidth(oBrowse:colCount) + nColSepWidth)
oBrowse:delColumn( oBrowse:colCount )
return nil

static function LoadBrow2(oBrowse, x)
local nCols
local oTempColumn
nRightCol := x
```



```

nColWidth := 0
//----- add columns and increment total cumulative column width
do while nColWidth <= nBrowseWidth
  oBrowse:addColumn( aColumns[x-] )
  nColWidth += oBrowse:colWidth(oBrowse:colCount) + nColSepWidth
enddo
nLeftCol := ++x
nColWidth := (oBrowse:colWidth(oBrowse:colCount) + nColSepWidth)
//----- lop off rightmost column in browse object
oBrowse:delColumn(oBrowse:colCount)
nCols := oBrowse:colCount
//----- because the columns in the TBrowse object are currently
//----- reversed, we must swap them to prevent a mirror image
for x := 1 to int(nCols / 2)
  oTempColumn := oBrowse:getColumn(nCols - x + 1)
  oBrowse:setColumn(nCols - x + 1, oBrowse:getColumn(x))
  oBrowse:setColumn(x, oTempColumn)
next
return nil

```

```

static function ClearBrowse(oBrowse)
//----- delete all columns from this browse object
do while oBrowse:colCount > 0
  oBrowse:delColumn(1)
enddo
return nil

```

الاستعراض Browse متعدد الصفوف (القسم الأول)

هو جدول استعراض يتيح لك نشر البيانات في سجل واحد على أكثر من صف واحد. إن هذه الإمكانية مثالية في الحالات التي تريد فيها عرض كافة المعلومات التي في سجل معين دون الاضطرار للانتقال أفقياً لمشاهدة الحقول "المخفية".

جوهر الأمر أننا نستطيع وضع أي شيء في العمود. وقد مرر معنا كيفية استعراض المصفوفات وحقول الذاكرة بإنشاء "مؤشر السجل" record pointer الخاص بنا.

```
c := TBColumnNew( "", { | | marray[curr_ele] } )
```

يقوم المتغير CURR_ELE بدور مؤشر في المصفوفة (أو بما تستعرض). ويمرر عادة بالإشارة إلى وظيفة "تجاوز كتلة" skipBlock ، بحيث يمكن معالجته لتابعة ومعرفة مكان المؤشر.

بعد هذا يجب علينا استخدام مصفوفة حرفية في العمود:

```
c := TBColumnNew( , { | | { field1, field2, field3, ;  
replicate(chr(196), 25) } [curr_ele] } )
```

للعמוד الآن خيار عرض واحدة من أربع قيم مختلفة: FIELD1 و FIELD2 و FIELD3 أو سطر أفقي (يعمل كفاصل بين السجلات). وستحدد قيمة CURR_ELE أيّاً من القيم الأربعة التي ستعرض.

إذا كنت تعرف مسبقاً عدد الحقول التي تريد استخدامها من بنية قاعدة البيانات ، فيمكنك استخدام وظيفة (FIELDGET) في كليبر 5 بدلاً من إنشاء مصفوفة حرفية.

ينبغي أن تعالج وظيفة "تجاوز كتلة" skipBlock "مؤشر CURR_ELE لتابعة وتحديد عنصر البيانات التي ستعرض. والتركيب هو:

```
b:skipBlock := { | skipCnt | gilligan(skipCnt, @curr_ele, max_ele, ;  
b:getColumn(b:colPos)) }
```

لقد تم تمرير SkipCnt داخلياً بواسطة TBrowse ، ليبدل على عدد الصفوف التي ستتحرك. ويمرر مؤشر العنصر (CURR_ELE) بالإشارة. بحيث يمكن أن تغيره وظيفة "تجاوز كتلة" skipBlock مباشرة ، ويجب أيضاً تمرير العدد الأقصى من

العناصر (MAX_ELE) لأن "تجاوز كتلة" skipBlock يفحص ويدقق مؤشر العنصر على ضوئه.

القاعدة المنطقية بسيطة: إذا كنت تتحرك إلى الأمام بتزايد CURR_ELE دون تحريك مؤشر السجل في إعادة البيانات. حينما تساوي قيمة MAX_ELE (إجمالي عدد الصفوف في كل سجل) يعاد ضبط CURR_ELE إلى (١) ، وتتجاوز (العدد المحدد) في قاعدة البيانات وتبدأ الحلقة الثانية. أما إذا كنت تتحرك إلى الخلف فالقاعدة معكوسة: يتناقص CURR_ELE حتى يصل إلى (١) حيث يعاد ضبطه إلى الحد الأقصى ، فتجاوز إلى الوراء وتبدأ ثانية من النهاية.

نقاط هامة

العدد الأقصى للعناصر: يجب تحديد قيمة قصوى لمؤشر العناصر. وفي المثال التالي ضبطنا المتغير MAX_ELE على عدد الحقول في قاعدة البيانات. وإذا أردت مثلاً الحقلين الأولين في قاعدة البيانات فيمكن تغييره بضبطه على (٢).

"الانتقال إلى أعلى" / "الانتقال إلى أسفل" goTop()/goBottom() : عندما نقفز إلى أعلى أو أسفل باستخدام الوظيفة "الانتقال إلى أعلى" (goTop()) أو "الانتقال إلى أسفل" (goBottom()) فيجب ضبط مؤشر CURR_ELE وفقاً لذلك. عند أعلى الملف يضبط CURR_ELE على (١) بحيث يعرض الحقل الأول بشكل صحيح. أما عند آخر الملف (في الأسفل) فيجب أن يضبط CURR_ELE على أقصى عدد من عناصر البيانات (MAX_ELE) لأننا سنكون في الصف الأخير من السجل الأخير. لقد ضمنا هذه القاعدة المنطقية مباشرة في حلقة ضغط المفاتيح الخاصة بالمفاتيح [PgUp] و [Ctrl]- [PgDn] و [Ctrl] ومع ذلك يمكنك القيام بكتابة "الانتقال لكتلة أعلى" goTopBlock و "الانتقال لكتلة أسفل" goBottomBlock كما يلي:

```
b:goTopBlock := { | | b:goTop( ), curr_ele := 1 }
b:goBottomBlock := { | | b:goBottom( ), curr_ele := max+ele }
```

إعادة الضبط بعد التأسيس: عندما تعرض البيانات في نافذة الاستعراض TBrowse لأول مرة سيضبط مؤشر العنصر الخاص بك على عنصر البيانات المعروض في الصف الأخير في الأسفل. ويجب 'إعادة ضبطه على (١) لأن عمود التظليل سيكون على الصف الأول. تعالج هذه الحالة القاعدة المنطقية التالية أسفل حلقة التأسيس:

```
if firstloop
  curr_ele := 1
  firstloop := .f.
endif
```

لتحديد البيانات في الخلية الحالية نحتاج إلى إشارة إلى العمود الحالي بحيث يمكننا تقييم كتلة الشيفرة الخاصة بها وهذا سهل وفعال.

```
// filename: TBROW44.PRG
#include "inkey.ch"
```

```
function tbrow44
local x
local nOldcursor := setcursor(0)
local cOldcolor := setcolor("+w/b")
```

```
//----- dummy data for purpose of this example
local aData := { { "Greg Lief", "Grumpfish, Inc.", "Salem, OR" }, ;
  { "Joe Booth", "CLIPWKS", "Philadelphia, PA" }, ;
  { "Mike Mussina", "Baltimore Orioles", "" }, ;
  { "Mary Gries", "Grumpfish, Inc.", "Salem, OR" }, ;
  { "Ira Emus", "Extrasensory Software", "Los Angeles, CA" }, ;
  { "Tom Glavine", "Atlanta Braves", "" }, ;
  { "Bill Gates", "Microsoft Corp.", "Redmond, WA" }, ;
  { "Cal Ripken", "Baltimore Orioles", "" }, ;
  { "Dave Rifkind", "Extrasensory Software", "Los Angeles, CA" } }
```

```
//----- create temporary database
dbcreate("tbrow44", { { "NAME", "C", 26, 0 }, ;
  { "COMPANY", "C", 26, 0 }, ;
  { "ADDRESS", "C", 26, 0 } } )
use tbrow44 new
for x := 1 to len(aData)
  append blank
  aeval(aData[x], { | ele, count | fieldput(count, ele) } )
next
scroll()
MultLine()
```



```

setcolor(cOldcolor)
setcursor(nOldcursor)
use
ferase("tbrow44.dbf")
return nil

// end main stub program
//-----//

/*
  Function: MultLine()
*/
static function MultLine
local x
local oBrowse := TBrowseDB(0, 0, maxrow(), maxcol())
local oColumn
local nKey := 0
local nCurrEle := 1
local nMaxEle
local IFixUp := .t.

oBrowse:skipBlock := { | nSkipCnt | Gilligan(nSkipCnt, @nCurrEle, ;
      nMaxEle, oBrowse:getColumn(oBrowse:colPos)) }
oBrowse:colorSpec := "+W/B, +W/R, W/G, +W/G"

nMaxEle := fcount() + 1
oColumn := TBColumnNew("Info", { || { tbrow44->name, ;
      tbrow44->company, tbrow44->address, ;
      replicate(chr(196), 25))[nCurrEle] } } )

//----- draw separators in different color than data
oColumn:colorBlock := { || if(nCurrEle == nMaxEle, {3, 4}, {1, 2}) }

oBrowse:AddColumn(oColumn)

//--- second column shows record # to prove that this beast works!
oColumn := TBColumnNew("Rec #", ;
      { || if(nCurrEle <> nMaxEle, recno(), ' ') } )
oBrowse:AddColumn(oColumn)
go top

//--- main keypress loop begins...
do while nKey <> K_ESC
  do while ( nKey := inkey() ) == 0 .and. ! oBrowse:stabilize()
    enddo

```



```

//----- if this is the first time in, we must reset element pointer
//----- which will have been torn asunder by the skipBlock below
if !FixUp
    nCurrEle := 1
    !FixUp := .f.
endif
if nKey == 0
    nKey := inkey(0)
endif
do case
    case nKey == K_UP
        oBrowse:up()
    case nKey == K_DOWN
        oBrowse:down()
    case nKey == K_CTRL_PGUP
        oBrowse:goTop()
        nCurrEle := 1
        !FixUp := .t.
    case nKey == K_CTRL_PGDN
        oBrowse:goBottom()
        nCurrEle := nMaxEle
endcase
enddo
return nil

// end static function MultLine()
//-----//

```

```

/*
Function: Gilligan()
Purpose: Custom skipblock function for MultLine()
*/
static function Gilligan(nSkipCnt, nEle, nMaxEle, oColumn)
local nMovement := 0
do case
    // no nMovement
    case nSkipCnt == 0
        skip 0
    // moving forward
    case nSkipCnt > 0
        do while nMovement < nSkipCnt .and. ! eof()
            if nEle < nMaxEle
                nEle++
                //----- if this cell is empty, skip past it without displaying
                if empty( eval(oColumn:block) )

```

```
        nEle++
      endif
    else
      skip 1
      nEle := 1
    endif
    nMovement++
  enddo
  if eof()
    go bottom
    nEle := nMaxEle
    nMovement--
  endif
  // moving backward
  case nSkipCnt < 0
    do while nMovement > nSkipCnt .and. ! bof()
      if nEle == 1
        nEle := nMaxEle
        skip -1
      else
        nEle--
        //----- if this cell is empty, skip past it without displaying
        if empty( eval(oColumn:block) )
          nEle--
        endif
      endif
      nMovement--
    enddo
    if bof()
      go top
      nEle := 1
      nMovement++
    endif
  endcase
  return nMovement

// end static function Gilligan()
//-----//
```

توجد نسخة ، يمكن أن تجدها في الملف TBROW45.PRG ، صالحة للعمل مع كل التحركات العمودية المطلوبة (← و → و PgUp و PgDn و Home و End). كما أنها تعرض لك كيف يمكنك عرض عدة حقول في عمود واحد ، بينما تشاهد

حقلا واحدا في عمود آخر. كما تعرض لك كيف يمكنك استعراض حقل مذكرة ،
بحيث يستطيع المستخدم من النظرة الأولى مشاهدة محتويات الحقل.

القسم الثاني

أهداف GET / نظام GET

تهيد

إن الغرض الرئيسي من أي برنامج لإدارة قاعدة البيانات هو تخزين البيانات في ملفات قاعدة البيانات. ولكن يجب أولاً إدخال هذه البيانات. تعرف هذه الطريقة لتخزين البيانات في أنواع XBase بأوامر GET. أن برمجة إدخال البيانات عملية تتم في خطوتين: يعرض المبرمج واحداً أو أكثر من أوامر GET على الشاشة ، يمثل كل منها عنصر بيانات يجب إدخاله ، ثم يصدر أمر "قراءة" READ الذي يشغل بدوره أوامر GET ويتيح للمستخدم إدخال صيغ مختلفة من البيانات فيها.

لا يوجد في أي لغة برمجة شيء يوفر مرونة أكثر من أوامر GET في كليب 5.x ، الذي يحتوي تصنيفات جديدة في GET ونظام GET يمكن إعادة تشكيله سنين في هذا البحث كيفية معالجة الهدف بواسطة الوظائف والمتغيرات الفورية الخاصة به. كما سنشرح طرقاً فعالة لاستخدام عبارة WHEN الجديدة ، وكيفية تحسين قوة أوامر "@.GET" و "READ".

بوجود هدف GET ونظام GET الذي يمكن إعادة تشكيله نستطيع تشكيل أوامر GET كما نرغب ، ونتحكم بها بشكل تام ، ويتوفر لنا إمكانية الوصول إلى عملية "القراءة" READ بصيغة وظيفة () READMODAL ووظائف مساندة ، بل إن هذه الوظيفة تم كتابتها بالكامل بلغة كليب (ومن السهل تعديله عند اللزوم). توجد شيفرة المصدر الخاصة بوظيفة () READMODAL في الملف GETSYS.PRГ الذي يوفره كليب.

تحليل أمر @..GET

هناك طريقتان لإنشاء أهداف GET ، أولهما ، وهي الشائعة أكثر: أن يقوم المعالج الأولي بهذا العمل بصيغة أمر @..GET. والثانية: باستخدام وظيفة كليبر GETNEW() (سنبحث هذه الطريقة بتفصيل أكثر لاحقاً). لفهم هذه العملية الهامة سنستعرض البنية اللغوية لأمر @..GET التي يحددها المستخدم:

```
// excerpted from the Clipper 5.2 version of STD.CH
#command @ <row> , <col> GET <var> [PICTURE <PIC>] ;
          [VALID <valid>] [WHEN <when>] => ;
      SetPos(<row> , <col> ; ;
      AAdd( GetList, _GET_(<var> , <(var)> , <pic> , ;
            <{valid}> , <{when}>) :display( ) )
```

سنكتب نموذجاً لبرنامج قصير يستدعي هذا الأمر ، ثم نفحص كل عبارة من أمر @..GET ، لنرى كيف يعمل المعالج الأولي بهذا الأمر. وسنتعرف فيما يلي على مؤشرات نتائج المعالجة الأولية ووظيفة كل منها.

الملف الاصيلي (.PRG):

```
@ 20, 0 get x picture '####' valid : empty(x) when y > 5
```

ملف المعالج الأولي (.PPO):

```
SetPos(20, 0) ; ;
AAdd( GetList, _GET_(x , "x", "####", { | | : empty(x) } , ;
      { | | y > 5 } ) :display( ) )
```

إن مواضع الصفوف والأعمدة مخرجات output تستخدم مؤشر النتائج العنادي لاستدعاء وظيفة "ضبط الموضع" SetPos() ، وهذا سيضع المؤشر في الصف (٢٠) في العمود (٠). وهذا هام لأن الموضع الحالي للمؤشر سيستخدم لتحديد موضع هدف

GET حال إنشائه. لذلك ينبغي نقل المؤشر إلى الموضع المطلوب قبل إنشاء هدف GET .

ثم يعالج المعالج الأولي كافة العبارات الأخرى لإنشاء استدعاء لوظيفة Clipper_GET_() الداخلية وإن اسم متغير GET مُخرج مرتين: مرة مع مؤشر النتائج العادي ، وأخرى مع مؤشر النتائج في الكتل:

GET(x , "x" , ...)

إن عبارة "صورة" PICTURE مُخرج مع مؤشر النتائج العادي حيث إنها بصيغة سلسلة حرفية:

GET(x, "x", "####", ...)

وإن عبارة "صحيح" VALID (بعد التدقيق) مُخرج مع مؤشر النتائج في الكتل ، لأن نظام GET سيتوقع أن تكون بصيغة كتلة شيفرة بحيث يمكنه تقييمها عند الضرورة:

GET(x, "x" , "####", { | | : empty(x) },)

وإن عبارة "WHEN" (قبل التدقيق) مُخرج مع مؤشر النتائج في الكتل. أيضاً مثل عبارة "صحيح" VALID سيتوقع نظام GET أن تكون بصيغة كتلة شيفرة بحيث يمكنه تقييمها عند الضرورة.

GET(x, "x" , "####", { | | : empty(x) }, { | | y > 5 })

إن هدف GET الذي تم إنشاؤه بواسطة وظيفة _GET_() يعرض بعد ذلك بواسطة وظيفة () get:display . وأخيراً يضاف هدف GET الجديد إلى مصفوفة تدعى GETLIST. والتي سنتكلم عنها بالتفصيل بعد قليل.

يجب ملاحظة أن وظيفة () _GET_ تنشئ أيضاً كتلة شيفرة تستخدم لمعالجة المتغيرات. تغيير أهداف GET قيم المتغيرات بواسطة تقييم كتلة الشيفرة هذه بدلاً من معالجة المتغيرات مباشرة. وعندما نتدخل قيماً في GET فإننا لا نغير المتغير ، بل ندخل

رموزاً وحروف في ذاكرة مؤقتة تمر لاحقاً كمتغيرات لكتلة الشيفرة هذه. وسنبحث بنية كتلة الشيفرة عند مناقشة المتغير الفوري "كتلة" block.

كما هي الحال مع موضع المؤشر الحالي ، عند إنشاء هدف GET يؤخذ بعين الاعتبار ضبط اللون الحالي. وستستخدم مجموعات الألوان الحالية غير المختارة عندما لا تختار هدف GET ، بينما تستخدم مجموعات الألوان المحسنة عند اختياره. فمثلاً ، إذا كانت مجموعة الألوان الحالية كما يلي:

```
setcolor("w/n, +w/r, , , w/b")
```

فسيعرض هدف GET باللون الأبيض على خلفية زرقاء عند عدم اختياره ، وباللون الأبيض الناصع على خلفية حمراء عند اختياره. يمكنك أيضاً تغيير المتغير الفوري "طيف الألوان" colorSpec المرتبط بهدف GET بعد إنشائه. كما يمكنك في الإصدار 5.01 من كليبر من استخدام العبارة الاختيارية "اللون" COLOR لتغيير "طيف الألوان" أثناء إنشاء هدف GET ، وسنبحث ذلك بالتفصيل أكثر لاحقاً.

عبارات @.GET في كليبر الجديد

ذكرنا أنه يمكننا تحديد لون هدف GET باستخدام العبارة الاختيارية "اللون" COLOR. وإذا أردنا استخدامها ، يجب أن تكون مجموعة الألوان بصيغة: <enhanced> , <standard>. يستخدم اللون المحسن <enhanced> عندما يظل هدف GET ، بينما يستخدم اللون الآخر عندما لا يظل الهدف. فعلى سبيل المثال: إذا أردت عرض هدف GET الخاص بك بلون أبيض على خلفية حمراء عندما يظل، وباللون الأبيض على خلفية زرقاء عندما لا يظل ، يمكنك استخدام القاعدة اللغوية بالطريقة التالية:

```
@ 20, 0 get x color "w/b , w/r"
```


يمكنك استخدام عبارة "أرسل" SEND لإرسال رسائل إلى أهداف GET. وسيتضح لنا استخدام هذه العبارة أكثر بعد بحث وظائف GET بتفصيل أكثر، فمثلاً: يستخدم ملف الترويسة STD.CH عبارة "أرسل" SEND كخطوة متوسطة لتنفيذ عبارة @..GET..COLOR:

الخطوة الأولى:

```
#command @ <row> , <col> GET <var>
                                [<clauses,...>]
                                COLOR <color>
                                [<moreClauses, ....> ]
=> @ <row> , <col> GET <var>
                                [<clauses>]
                                SEND colorDisp(<color>)
                                [<moreClauses>]
```

الخطوة الثانية:

```
#command @ <row> , <col> GET <var>
                                [PICTURE <pic>]
                                [VALID <valid>]
                                [WHEN <when>]
                                [SEND <msg>]
=> SetPos( <row> , <col> ) ;
AAdd(GetList, ;
_GET_(<var> , <var> , <pic> , <{valid}> , <{when}> ) :display( ) ) ;
[ ; ATail(GetList) : <msg> ]
```

ميزة التلوين التلقائي

يمتاز كليبر 5.x عن كليبر Summer'87 ، أنه يقوم بإعادة تلوين أهداف GET تلقائياً بعد وظيفتي WHEN و "صحيح" VALID ، ونحتاج للقيام بذلك يدوياً.

مصفوفة GETLIST

إذا كنت تجمع برامج كليبر 5.2 بواسطة خيار الجمع (/W) فلا بد أن تكون قد عرفت مصفوفة GETLIST المستخدمة بكثرة في برامج كليبر 5.2 كميزة عن إصدارات كليبر السابقة. وهذا ضروري لأنه من المحتمل كتابة كتلة الشيفرة التالية (إذا لم يسبق لك وأن كتبت شيفرة بهذا الشكل ، فلابد الآن):

```
function 1
@ 1, 1 get x
2( )
read
return nil
function 2
@ 2, 1 get y
3( )
return nil
.
function 3
@ 3, 1 get z
return nil
```

ستكون أهداف GET الثلاثة هذه بحالة تشغيل عند إدخال أمر القراءة READ. وإذا لم تكن مصفوفة GETLIST مرئية في كافة أجزاء البرنامج فستؤدي طريقة التشفير هذه إلى توقف برنامج DOS عن العمل مباشرة.

كلما أصدرنا أمر GET.@ يترجم المعالج الأولي إلى قيمة منطقية تنشئ هدف GET جديد وتضيفه إلى مصفوفة GETLIST. وعندما نصدر أمر القراءة READ لاحقاً تمرر محتويات GETLIST إلى وظيفة (READMODAL) التي تقوم بعد ذلك بتأسيس عملية تعديل وتحرير الشاشة بأكملها. وبذلك يكون كل هدف من أهداف GET عنصراً في مصفوفة GETLIST ، وينتقل نظام GET بينها عندما تضغط مفاتيح الانتقال (والتي يمكنك بكل سهولة إعادة تعريفها إذا رغبت في ذلك).

عند انتهاء عملية القراءة READ تعيد وظيفة (READMODAL) التحكم إلى برنامج الاستدعاء ويفرغ مصفوفة GETLIST لاستخدامه ثانية (مالم تخصص خيار (READ SAVE).

عمليات القراءة المتداخلة Nested Reads

لإجراء عمليات القراءة المتداخلة نحتاج أن نستفيد من الاعلان "محلي" LOCAL بواسطة إنشاء مصفوفة GETLIST "محلي" LOCAL في كل إجراء يستخدم أوامر GET. نستخدم في المثال التالي القراءة المتداخلة في الوظيفة valid التي يتم استدعاؤها من الأمر GET الثاني.

```
// filename: GETS01.PRG
function gets01
local a := 0
local b := 0
local c := 0
scroll()
@ 1,1 get a
@ 2,1 get b valid nestread()
@ 3,1 get c
read
return nil

static function nestread
local d := 1
// for a nested read, all you need to do is put the following line
// in the function where the nested read will take place (and be
// sure not to use the CLEAR GETS command)
local getlist := {}
@ 5, 1 say "In nested read..."
@ 6, 1 get d
read
scroll(5, 0)
//
// If you are using Clipper 5.2 and want to terminate the upper-level
// READ based on what was entered here, insert the following statement
// here:
//
// ReadKill(.T.)
//
return .t.
```

تحذير

لن تحتاج بعد ذلك لاستخدام أمر CLEAR GETs ، بل ينبغي ألا تستخدمه لأن له تأثيراً جانبياً سيئاً في "مسح" كافة مستويات GETs.

استهلال أمر GET (كليبر 5.2 فقط)

يوفر الإصدار الجديد من كليبر 5.2 ، الوظيفة (ReadModal) ، والتي تقوم بتمرير متغير رقمي ثان لبيان أمر GET الذي يجب تشغيله أولاً. ولكن لا يمكن الاستفادة من ذلك مع الأمر المعتاد READ ، لذلك يفضل إدراج الأوامر التالية في برنامجك أو في أحد ملفات الترويسة الخاصة بك:

```
#xcommand READ INITIAL <init>
                                => :
                                :
                                :
                                ReadModal( getlist, <init> )
                                ; getlist := { }

#xcommand READ SAVE INITIAL <init>
                                => :
                                :
                                :
                                ReadModal( getlist, <init> )
#xcommand READ INITIAL <init> SAVE
                                => :
                                :
                                :
                                ReadModal( getlist, <init> )
// quick example

function main
local x := 1
local y := 2
local z := 3
scroll ( )
ER TEXT = read initial 2
return nil
```


تحسين نظام إدخال البيانات

الطريقة الأولى: بدائية

عند إنشاء شاشة إدخال بيانات "فإن الطريقة الجيدة قديماً" كانت أن تصدر أوامر GET مع أمر القراءة READ ضمن حلقة DO WHILE. وهذا يتيح للمستخدم فرصة تأكيد ما حره أو عدله ، أو أن يعيد إدخاله إذا لزم الأمر. ونصدر أوامر @.GET كلما أردنا تحرير البيانات أو تعديلها.

```
function test
local a := 0
local b := 0
local c := 0
local IMainloop := .t.
local getlist := { }
local nChoice
do while IMainloop
@ 0,0 get a
@ 1,0 get b
@ 2,0 get c
read
nChoice := alert("What's next, Kemosabe?", ;
                { "Save" , "Re-edit", "Abort" } )
IMainloop := (nChoice == 2 )
enddo
if nChoice == 1
    // stick the data into the database fields
endif
return nil
```

بهذه الطريقة ، نقوم بإنشاء أهداف GET داخل حلقة DO WHILE ، ثم ننهاها. ونقوم بإعادة إنشائها في كل مرة نريد تحرير البيانات أو تعديلها.

الطريقة الثانية: أفضل (GETLIST)

بدلاً من إصدار أوامر @.GET ضمن حلقة DO WHILE ، سنضعها قبل الحلقة. أي أن أهداف GET تحتاج فقط إلى إنشاء مرة واحدة فقط. ونستدعي وظيفة ReadModal() مباشرة ضمن الحلقة بدلاً من إصدار أمر القراءة READ. وبذلك لا تُمسح مصفوفة GETLIST في كل مرة ، نخرج من أمر القراءة READ ، كما أنه لا حاجة لإعادة إنشاء أهداف GET في كل مرة . ولاداعي لمسح مصفوفة GETLIST في نهاية الوظيفة. وبما أنه محدد كمحلي "LOCAL" فسيختفي من تلقاء ذاته.

```
function test
local a := 0
local b := 0
local c := 0
local lMainloop := .t.
local getlist := { }
local nChoice
@ 1, 0 get b
do while lMainloop
    readmodal(getlist)
    nChoice := alert("What's next, Kemosabe?", ;
                    { "Save" , "Re-edit", "Abort" } )
    lMainloop := (nChoice == 2 )
enddo
if nChoice == 1
    // stick the data into the database fields
endif
return nil
```

وعلى الرغم من أن هذه الطريقة أكثر احترافاً من الطريقة السابقة ، إلا أنه لازال في وسعنا تطوير هذه الطريقة إلى الأفضل.

الطريقة الثالثة: الأفضل (STATIC GETLIST)

في مثالنا الأخير ، أصدرنا أوامر GET مرة واحدة فقط ، وذلك قبل حلقة DO WHILE. يمكننا تحسين هذه الطريقة بإعلان مصفوفة GETLIST على أنها من النوع الساكن STATIC. أي أننا ننشئ أهداف GET مرة واحدة فقط مهما تعددت المرات التي نستدعي فيها وظيفة إدخال البيانات.

ويجب أيضاً تحديد نطاق المتغيرات المرتبطة بأهداف GET على أنها ساكنة ، لكي نضمن مستوى الرؤية ذاته في المتغيرات وأهداف GET. ولهذا السبب قد نحتاج إلى إعادة ضبط قيم المتغيرات في كل مرة نستدعي الوظيفة ، وإلا فستحتفظ بقيمتها السابقة مما سيسبب الإرباك للمستخدم. ومع ذلك تبقى هذه الطريقة أفضل من إعادة تجهيز أهداف GET كلما إستدعينا الوظيفة ، خاصة إذا كان لدينا عدد كبير.

ملاحظة

بما أننا نغير قيم المتغيرات المرتبطة بأهداف GET ، يجب أن نطلب من الأهداف أن تعيد عرض ذاتها (باستخدام الوظيفة (get:display)). وهذا يضمن إظهار القيم المعدلة في الشاشة لمنع إرباك المستخدم.

```
function test
static a := 0
static b := 0
static c := 0
local getlist := { }
static IMainloop := .t.
local nChoice
local nGets
local n
if empty(getlist)
    // first time in: create the GET objects
    // (we only have to do this once, no matter
    // how many times we call this function!)
```



```
@ 0, 0 get a
@ 1, 0 get b
@ 2, 0 get c
else
    // one subsequent visits, we re-initialize the values
    // of the variables ....no need to read the @..GETs
a := b := c := 0

    // tell Get objects to redisplay themselves so that
    // the new values are reflected on the screen
nGets := len(getlist)
for n := 1 to nGets
    getlist[n]:display( )
next
endif

do while IMainloop
    readmodal(getlist)
    nChoice := alert("What's next, Kemosabe?", ;
                    { "Save" , "Re-edit", "Abort" } )
    IMainloop := (nChoice == 2 )
enddo
if nChoice == 1
    // stick the data into the database fields
endif
return nil
```

إذا أردنا أن تحتفظ أوامر GET بقيمتها السابقة كلما دخلنا إلى شاشة إدخال البيانات يمكننا حذف الشيفرة التي تزيل القيم من الشاشة وتلف ضمن مصفوفة GETLIST لإعادة عرض أوامر GET.

عبارة WHEN

مرّ معنا عبارة "صحيح" VALID التي توفر تدقيقاً لاحقاً post-validation لكل أمر GET وتمنع الخروج في حال عدم توفر شروط معينة. أما عبارة "WHEN" فتوفر تدقيقاً سابقاً pre-validation ، وعند تحديدها ستقيم قبل دخول أمر GET ، فإن عادت قيمتها "غير حقيقية" False فستمنع الدخول لأمر GET.

تجاوز أمر GET

يبين المثال التالي استخدام عبارة "WHEN". لن يتمكن المستخدم من إدخال رقم بطاقة الائتمان ما لم يضبط متغير رصيد الدائن إلى "حقيقي" True.

```
// filename: GETS02.PRG
function gets02
local fname := space(15), lname := space(15), credit := .f.
local cardno := space(20), custno := space(6)
local getlist := {}
scroll()
@ 10, 20 say "First Name: " get fname
@ 11, 20 say "Last Name: " get lname
@ 12, 20 say "Credit? " get credit
@ 13, 20 say "Card Number:" get cardno when credit
@ 14, 20 say "Customer No:" get custno
read
return nil
```

توفر عبارة "WHEN" إمكانيات عديدة باستدعاء الوظائف منها. كما هو الحال مع عبارة صحيح VALID ، فما عليك إلا التأكد من أن عبارة "WHEN" تقيم بقيمة منطقية (يجب أن تكون هذه القيمة "حقيقية" True إذا أراد المستخدم إدخال أمر GET ذاك).

رسائل المساعدة باستخدام عبارة WHEN

يبين المثال التالي كيفية توفير رسالة لكل أمر GET.

```
// filename: GETS03.PRG
function gets03
local name, address, city, getlist := {}
scroll()
dbcreate('temp', { { 'name', 'C', 20, 0 } , ;
                    { 'address', 'C', 25, 0 } , ;
                    { 'city', 'C', 20, 0 } } )
use temp new
append blank
scroll()
name := temp->name
address := temp->address
city := temp->city
@ 10,0 get name when fieldhelp(24, 1, "Please enter a name")
@ 11,0 get address when fieldhelp(24, 1, "Please enter an address")
@ 12,0 get city when fieldhelp(24, 1, "Please enter a city")
read
temp->name := name
temp->address := address
temp->city := city
use
ferase('temp.dbf')
return nil

function fieldhelp(row, col, msg)
@ row, col say padr(msg, 50)
return .t.
```

إدخال البيانات باستخدام WHEN

إذا كنت تستخدم "مكتبة جرمفيس" Grumpfish Library فلا بد أن تكون قد مررت معك وظيفة () APICK التي تختار بسهولة من أحد مصفوفات الاختيارات المتوفرة. وربهطه مع عبارة "WHEN" يصبح أداة قوية وفعالة تمكن المستخدم من اختيار قيمة بدلاً من إدخالها. وتوضح القائمة التالية هذا المبدأ. فإذا لم تكن تستخدم "مكتبة

جزمفيش " فاستبدال وظيفة () APICK بوظيفة () ACHOICE. ومع ذلك ، يمكنك كتابة وظيفة تستدعي وظيفة () ACHOICE وتقوم بمسح الشاشة بأكملها (كما تفعل وظيفة () APICK).

```
// filename : GETS04.PRG
function gets04
local days := { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" }
local mday := space(9), mmonth := space(9)
local months := { "January", "February", "March", "April", "May", ;
                  "June", "July", "August", "September", "October", ;
                  "November", "December" }
local getlist := {}
scroll()
@ 0, 10 say "Day: " get mday when ;
    empty(mday := days[ max(apick(9, 34, 15, 44, days), 1) ])
@ 1, 10 say "Month:" get mmonth when ;
    empty(mmonth := months[ max(apick(8, 34, 16, 44, months), 1) ])
read
return nil
```

لاحظ التركيب الدقيق لعبارة WHEN. تعيد وظيفة () APICK قيمة رقمية تتوافق مع عنصر المصفوفة المختار. فإذا ضغط المستخدم مفتاح [Esc] لمغادرة وظيفة () APICK فستعيد القيمة "صفر". وبما أن مصفوفات كليبر تعتمد على قيمة "الواحد" بدلاً من قيمة "الصفر" ، فإن الإشارة إلى عنصر المصفوفة (صفر) يكون باتجاه واحد إلى نظام التشغيل DOS. لهذا يجب حذف هذه الإمكانية مع وظيفة () MAX التي تضمن أن يكون عنصر المصفوفة الأدنى المشار إليه واحداً. ثم نقوم بتعيين قيمة عنصر المصفوفة المناسب للمتغير ، وفحصه للتأكد من كونه فارغاً. وبما أنه لن يكون فارغاً ستعيد عبارة WHEN القيمة "غير حقيقي" False فتمنع المستخدم من إجراء تعديل آخر لأمر GET.

استخدام المفاتيح السريعة مع WHEN

باستخدام عبارة WHEN ووظيفة "ضبط المفاتيح" (SETKEY). يمكن ضبط مفاتيح مختلفة للاستخدام السريع لكل أمر من أوامر GET.

يبين المثال التالي قدرة عبارة WHEN ، وتصريح "ساكن" STATIC ، ووظيفة "ضبط المفاتيح" (SETKEY) على توفير الحلول المناسبة. فإذا مررت قيمة INKEY وكتلة شيفرة إلى وظيفة (HOTKEY2) (كما هو مشاهد في عبارات WHEN) سيتم ربط كتلة الشيفرة بذلك المفتاح بالذات. وإذا لم تمرر أي شيء (كما هو مشاهد في عبارات "صحيح" VALID) فسيعاد ضبط المفتاح الحالي إلى حالته السابقة.

```
// filename: GETS05.PRG
#include 'inkey.ch'

function gets05
local x := "Press F1 (1st hotkey)"
local y := "Press F2 (2nd hotkey)"
local z := "Press F3 (3rd hotkey)"
local getlist := {}
scroll()
@ 1,1 get x when hotkey2(K_F1, { || func1() }) valid hotkey2()
@ 2,1 get y when hotkey2(K_F2, { || func2() }) valid hotkey2()
@ 3,1 get z when hotkey2(K_F3, { || func3() }) valid hotkey2()
read
return nil

static function hotkey2(nkey, block)
static oldkey, oldblock
if oldkey == NIL // if we are entering this for the first time
    oldblock := setkey(nkey, block)
    oldkey := nkey
else
    setkey(oldkey, oldblock)
    oldkey := oldblock := NIL
endif
return .t.
```



```
static function func1
@ 24,0 say "This is the 1st hot key"
inkey(0)
@ 24,0
return nil
```

```
static function func2
@ 24,0 say "This is the 2nd hot key"
inkey(0)
@ 24,0
return nil
```

```
static function func3
@ 24,0 say "This is the 3rd hot key"
inkey(0)
@ 24,0
return nil
```

استخدام GETLIST مع عبارة WHEN

تمرر الشيفرات التالية مصفوفة GETLIST إلى وظيفة WHEN التي تغير كل أمر من أوامر GET لي مطابق الأمر الذي غادرته لتوك. فعلى سبيل المثال ، إذا غيرت GET رقم (١) إلى (١٠٠) فإن أوامر GET ذات الأرقام من (٢-٥) ستخصص لقيمة (١٠٠).
نلاحظ أن عبارة WHEN تكتب وكأنها كتلة شيفرة code block.

```
// filename: GETS06.PRG
function gets06
local a := { 1, 2, 3, 4, 5 }, getlist := {}
scroll()
@ 1,0 get a[1] color "W/B,W/R"
@ 2,0 get a[2] color "W/B,W/R" when redisplay(getlist, 1)
@ 3,0 get a[3] color "W/B,W/R" when redisplay(getlist, 2)
@ 4,0 get a[4] color "W/B,W/R" when redisplay(getlist, 3)
@ 5,0 get a[5] color "W/B,W/R" when redisplay(getlist, 4)
read
return nil
```

```
function redisplay(gets, ele)
local value := gets[ele]:varGet()
local x
local y := len(gets)
dispbegin()
```

```
for x := ele + 1 to y
  gets[x]:varPut(value)
  gets[x]:display()
next
dispend()
return .t.
```

تعميم الشيفرة الخاصة بك

ورد في المثال السابق ثلاث طرق خاصة بفئة أهداف GET ، وهي: (varGet) و (varPut) ، ووظيفة العرض (display) . تسرجع وظيفة (get:varGet) القيمة الحالية لأمر GET ، بينما تعين (get:varPut) قيمة الأمر GET ، وتجعل (get:display) أمر GET يعيد عرض ذاته. يمكنك هذه الطرق الثلاث من كتابة شيفرة عامة كما سيتبين لنا في المثالين التاليين.

المثال الأول: الحصول على متغيرات محلية

على الرغم من أن كليبر 5.x لازال يحتفظ بوظيفة READVAR() ، إلا أننا لن نحتاج لاستخدامها ، لأنها لن تعمل على متغيرات "محلية" LOCAL و"ساكنة" STATIC (لأنها ليس فيها إدخالات جداول الرموز). يوضح المثال التالي كيفية معالجة أمر GET الحالي دون معرفة اسمه.

```
local x := 0
@ 1, 1 get x valid validate( )

function validate
local g := getactive( )
// present pick list from lookup database
// user makes selection
g:varPut( lookup_value )
return
```

مع أن وظيفة (GETACTIVE) (الموجودة في برنامج GETSYS.PRG) صغيرة جداً إلا أنها قوية جداً. فهي تعيد إشارة إلى هدف GET العامل حالياً. وحالما وجدت هذه الإشارة لديك ، يمكنك أن تفعل ما تريد بالهدف GET ، بما في ذلك تحريكه أو تغيير لونه أو صورته ، أو تغيير قيمته ، إلخ.

ذكرنا أعلاه أن وظيفة `get:varPut()` تعين قيمة لأمر `GET` ، فإذا أردنا معرفة قيمة هدف `GET` الحالي ، يمكننا استرجاعها بواسطة الوظيفة `get:varGet()` . ولن نحتاج مع هذه الوظائف ، إلى تمرير متغير ثانية بواسطة الإشارة إلى وظيفة "صحيح" `VALID`.

المثال الثاني: القراءة المتداخلة مع وظيفة `GETACTIVE()`

يبين المثال التالي إمكانية تغيير قيمة هدف `GET` الحالي ضمن عملية "قراءة متداخلة" `READ`. لاحظ أننا نستخدم وظيفة `GETACTIVE()` لمعالجة هدف `GET` بدلاً من تمريره بالإشارة أو بأية طريقة أخرى ، لاحظ أيضاً استخدام وظيفة `READKILL()` (المتوفر في كليبر 5.2 فقط) ، فهو يمكننا من إنهاء القراءة على مستوى أعلى في حال كون حساب الدائن أكبر من الرصيد المستحق.

```
// filename : GETS07.PRG
function gets07
local getlist := {}
local x := 0, mdate := date() + 14
scroll()
@ 10, 10 say "Balance: " get x picture '#####.##' valid credit()
@ 11, 10 say "Due date:" get mdate
setcursor(1)
read
setcursor(0)
return nil

static function credit
local g := getactive()
local x := 0
local getlist := {}
local oldvalue := g:varGet()
local oldscrm := savescreen(10, 40, 10, 64)
@ 10, 40 say "Credit (if any):" get x picture '#####.##'
read

#ifdef CLIPPER52

// if credit is larger than original balance, set balance to zero
// and terminate the upper-level READ
if x > oldvalue
```

```
g:varPut(0)
readkill(.t.)
else
  g:varPut( oldvalue - x ) // subtract credit from original balance
endif

#else

g:varPut( oldvalue - x ) // subtract credit from original balance

#endif
restscreen(10, 40, 10, 64, oldscm)
return .t.
```


حفظ أوامر GET باستخدام "المخزن" Stack

ينبغي أن تكون الآن قد أصبحت على معرفة جيدة ، لفهوم مدى الملف الساكن -file wide statics المرتبط بحفظ واسترجاع الوظائف. نستخدم في وظائف "مكتبة جرمفيس" Grumpfish library التالية ، منطق "stack" لحفظ واسترجاع أوامر .GET

```
// filename: SAVEGETS.PRG
static getstack_ := {}

/*
  GFSaveGets(<aGets>)
  Save the gets in <aGets> array on stack
*/
function GFSaveGets(getlist)
aadd(getstack_, getlist)
return len(getstack_)

/*
  GFRestGets( @<aGets> [, <nEle>]):
  Restore gets to <getlist> array which should be passed by reference
  Optional parameter <nEle> indicates which set of GETs to pull from
  stack. If not passed, LIFO logic will be used.
*/
function GFRestGets(getlist, ele)
if ele == NIL
  ele := len(getstack_)
endif
// preclude empty array
if ele > 0
  /* pull GETs from last element in array */
  getlist := getstack_[ele]
  /* truncate length of array only if using LIFO */
  if pcount() == 0
    asize(getstack_, ele - 1)
  endif
endif
return nil
```

تفيدك هذه الوظائف كثيراً إذا أردت إنشاء تراكيب إدخال بيانات في شاشات متعددة. فتمكنك من عرض وإخفاء أوامر GET الفعالة حسب رغبتك.

لاحظ أن مصفوفة `getstack` ستقطع من طرفها ، فقط إذا استخدمنا طريقة "آخر سجل مدخل هو أول سجل يسرجع" والتي يرمز لها بـ: (LIFO). أما إذا استخدمنا الوصول العشوائي مع وظيفة `GFRestGets()` بتمرير المتغير التالي ، فستفرض الوظيفة أنك لا تريد قطع آخر مجموعة من أوامر GET.

توضح الشيفرة التالية استخدام هذه الوظائف لإنشاء ثلاث مجموعات متوازنة من أوامر GET ودفع كل منها إلى مكس GET باستخدام الوظيفة `GFSaveGets()` ، ثم عرضها بترتيب مختلف مع وظيفة `GFRestGets()`. تعيد وظيفة `REGET()` عرض كافة أوامر GET العاملة مع وظيفة `get:display()`.

```
// filename: GETS08.PRG
function gets08
local getlist := {}
local x := { 1, 2, 3, 4, 5, 6, 7, 8, 9}, y, z
for z = 1 to 3
  for y = 1 to 3
    @ y * 2, 0 get x[(z - 1) * 3 + y]
  next
  gfsavegets(getlist)
  getlist := {}
next
scroll()
gfrestgets(@getlist, 2)
reget(getlist)
read
gfrestgets(@getlist, 1)
reget(getlist)
read
gfrestgets(@getlist, 3)
reget(getlist)
read
aeval(x, { | a | qout(a) } )
return nil
static function reget(gets)
aeval(gets, { | get | get:display() } )
return nil
```

شاشات إدخال بيانات متعددة الصفحات

بجئنا آنفاً أن أوامر GET تحمل في مصفوفة Getlist وتقرر إلى وظيفة Readmodal(). ويمكننا بهذه الطريقة إنشاء وتنفيذ شاشات إدخال بيانات متعددة الصفحات بفعالية عجيبة.

قبل صدور كليبر 5.0 ، كان يتطلب إنشاء شاشات إدخال بيانات متعددة الصفحات إعادة إصدار أوامر GET..@ في كل مرة تنتقل فيها بين الشاشات ، وقد كان هذا ممكناً لكنه بطيء جداً.

يمكننا التخلص من هذا الإجراء بتحميل عدة مصفوفات في أهداف GET. ثم لختار المصفوفة الموافقة ، حيثما تنتقل بين صفحات إدخال البيانات ، ونمررها إلى وظيفة Readmodal() .

نستخدم في المثال التالي ثلاث صفحات إدخال بيانات تحتوي كل منها على ٢٠ أمر GET تتوافق مع ٢٠ عنصر مصفوفة. وستخزن هذه الصفحات في مصفوفة _PAGES. ولأننا نحتاج أيضاً لإعادة تلوين أجزاء مختلفة من النص الساكن المرافق لأوامر GET هذه ، فسنحفظ صور الشاشات لكل صفحة إدخال بيانات في مصفوفة _SCREENS.

لإنشاء كل شاشة ، ننفذ ٢٠ أمراً من أوامر GET..@ لتحميل مصفوفة GETLIST مع ٢٠ هدف GET. ثم نقل أوامر GET هذه في العنصر الملائم من مصفوفة _PAGES (لاحظ وظيفة ACLONE() مما يضمن عدم تأثير أي تغيير لاحق بمصفوفة GETLIST في مصفوفة _PAGES). وتحفظ أيضاً الشاشة الحالية وتخزينها في العنصر الملائم في مصفوفة _SCREENS. ثم نمسح الشاشة ومصفوفة GETLIST ، ونكرر العملية.

يسبق هذه العملية وظيفة () DISPBEGIN ويلحقها وظيفة () DISPEND وهذا يجعلها غير مرئية بالنسبة للمستخدم ، لئلا يربكه ظهور الأوامر في الشاشة أمامه.

يلي ذلك حلقة التنفيذ. ولعرفة أية شاشة من شاشات GET فعالة حالياً نستخدم مؤشراً ليبن ذلك. تسرجع هذه البيانات في مصفوفة _SCREENS ، ويعاد عرض العنصر الملائم من مصفوفة _PAGES ثم يمرر إلى وظيفة () Readmodal . بعد انتهاء هذا الوظيفة ، نفحص مفتاح الخروج لتحديد الصفحة التي ستكون عاملة من بين صفحات GET. سينقلنا مفتاح [PgDn] (أو السهم إلى أسفل في آخر أمر GET) عبر الشاشات باتجاه الأمام ، بينما ينقلنا مفتاح [PgUp] (أو السهم إلى أعلى في أول أمر GET) إلى الخلف. وسينهي أي مفتاح آخر عملية القراءة READ.

```
// filename: GETA09.PRG
#include "inkey.ch"
```

```
function gets09
local a[60]
local x
local y
local nKey
local nPtr
local mainloop := .t.
local getlist := {}
local pages_[3]
local screens_[3]
local lOldreadexit := readexit(.t.)
//----- fill test array
for x := 1 to 60
    a[x] := x
next

dispbegin()
scroll()

for x := 1 to 3

    //----- create GET objects for this screen
    @ 0, 28 say "You are now viewing page " + str(x, 1)
    for y := 1 to 20
```



```

        @ y + 1, 1 say "This is GET #" + str(y + (x - 1) * 20, 2) + ":" ;
            get a[y + (x - 1) * 20]
    next
    @ maxrow(), 16 say "PgUp = prev page  PgDn = next page  ^W = save"

    //----- dump these GETs into the PAGES_ array and save current screen
    pages_[x] := aclone(getlist)
    screens_[x] := savescreen()

    //----- clear GETLIST array and screen
    getlist := {}
    scroll()
next

dispend()

//----- proceed with main loop
nPtr := 1
do while mainloop
    restscreen(,,,screens_[nPtr])
    aeval(pages_[nPtr], { | g | g:display() } ) // to redisplay all active GETs
    readmodal(pages_[nPtr])
    nKey := lastkey()

    //----- determine which GET screen should be displayed
    do case
        case nKey == K_PGUP .or. nKey == K_UP
            if nPtr == 1
                nPtr := 3
            else
                nPtr--
            endif
        case nKey == K_PGDN .or. nKey == K_DOWN
            if nPtr == 3
                nPtr := 1
            else
                nPtr++
            endif
        otherwise // any other key causes exit from loop
            mainloop := .f.
    endcase
enddo
readexit(IOldreadexit)
return nil

```


الوظيفة GETNEW()

تقوم الوظيفة GETNEW() بإنشاء هدف GET دون استخدام أمر GET..@. والقاعدة اللغوية لوظيفة GETNEW() إلى حد كبير ، ولكن ليس تماما ، مايقوم المعالج الأولي بإرساله إلى وظيفة GET_() الداخلية:

GetNew([<row> , [<column>] , [<block>] , [<var>] , [<picture>] , [<color>])

حيث أن "الصف" <row> و "العمود" <column> هما متغيران رقميان يمثلان موضع صف وعمود البداية في أمر GET على الشاشة.

أما الـ "كتلة" <block> فهي كتلة شيفرة للمتغير المطلوب. ويمكن تغيير قيمة هذا المتغير بواسطة كتلة الشيفرة هذه. أما <var> فهو رمز يمثل اسم متغير GET.

وأما "صورة" <picture> فهي رمز يمثل عبارة "الصورة" PICTURE التي ستستخدم لأمر GET. فإذا لم تمررها ، سيتم استهلاكها على أنها "صفر" NIL.

أما متغير "اللون" <color> فهو رمز يمثل ضبط اللون المستخدم لأمر GET. فإذا لم تمرره ستستخدم مجموعات الألوان غير المختارة و المحسنة.

جميع هذه المتغيرات السابقة اختيارية. فيمكنك تزويد أي منها أو جميعها إلى وظيفة GETNEW() ، أو يمكنك تعيينها فيما بعد.

أما الاختلافات الجوهرية بين وظيفة GETNEW() ووظيفة GET_() هي:

■ تقبل وظيفة GET_() ضبط DELIMITER ، في حين أن GETNEW لا تقبله.

- يمكنك الوظيفة GETNEW() من تموير ضبط الألوان كمتغير ، بينما تستخدم الوظيفة _GET_ ضبط اللون الحالي فقط.
- تنشئ الوظيفة _GET_() كتلة الشيفرة لمتغير GET تلقائياً ، ولا يقوم الأول بذلك.
- تقبل الوظيفة _GET_() كتل الشيفرة لعبارة (WHEN) قبل التدقيق وما بعد التدقيق (VALID) كمتغيرات. وإذا أردت استخدامها مع هدف GET تم إنشاؤه بواسطة الوظيفة GETNEW() فيجب عليك تعديل المتغير الفوري بعد ذلك.
- تعين الوظيفة _GET_ المتغير الفوري: get:subscript ، بينما لاتفعل الوظيفة GETNEW() ذلك. ولكن يمكن تعيين get:subscript في كليب 5.2 ، والذي يمكن أن يعالج هذا الاختلاف غير الهام. حتى لو كنت تستخدم كليب 5.0x فإن الوظيفة GETNEW() تتيح لك إدخال ما تريد في المتغير الفوري get:name.

المتغيرات الفورية لهدف GET

غالباً ما تواجهنا حالات نحتاج فيها إلى الرجوع إلى أحد المتغيرات الفورية المدخلة والخاصة بفئة هدف GET. يشتمل الجدول التالي على كافة المتغيرات الفورية لأوامر GET. ويمكنك إعادة تعيين المتغيرات المؤشرة بنجمة ("*").

النوع	الغرض	الاسم
L	فحص ذاكرة التعديل المؤقتة من أجل التاريخ غير الصحيح	badDate
B	كتلة الشيفرة التي تربط GET بالمتغير	block*
C	سلسلة حرفية تحتوي على الذاكرة المؤقتة للتعديل	buffer*
=	متغير معرف من قبل المستخدم "الشحنة"	cargo*
L	افحص إذا كانت الذاكرة المؤقتة ل: GET حصل عليها تعديل.	changed*
N	رقم عمود GET	col*
C	الألوان الخاصة ب: GET	colorSpec*
N	موضع الفاصلة العشرية داخل ذاكرة التعديل المؤقتة	decPos
N	حالة الخروج	exitState*
L	هل GET الحالية مظلمة؟ (وهذا يعني أن عليها تركيز إدخال)	hasFocus
C	اسم متغير GET	name*
C	سلسلة حرفية تحتوي على القيمة الأصلية ل: GET	original
C	سلسلة الصورة	picture*
N	موضع المؤشر الحالي داخل الذاكرة المؤقتة للتعديل	pos*
B	كتلة شيفرة لتصحيح القيمة المدخلة حديثاً	postBlock*
B	كتلة شيفرة تحدد ما إذا كان التعديل مسموح به أم لا	preBlock*
B	كتلة شيفرة محسنة تحدد كيف تم تعديل GET	reader*
L	رفض آخر عملية اقحام أو كتابة فوقية	rejected
N	رقم الصف "السطر"	row*
A	ترجع قيمة الرمز الفرعي بالنسبة للمصفوفة (فقط كليب 5.2)	subscript*
C	نوع متغير GET	type
L	عندما يحاول المستخدم الخروج من الذاكرة المؤقتة	typeOut

وكما هي الحال مع المتغيرات الفورية ، عند الإشارة إليها ، يجب أن تسبقها بالإشارة إلى هدف GET ، فمثلاً:

```
get:colorSpec := "w/r, w/b" //good
colorSpec := "w/r, w/b" // bad!
```

المتغير الفوري "تاريخ غير صحيح" badDate

يحتوي هذا المتغير الفوري قيمة منطقية تكون عادة "غير حقيقية" (F.). اللهم إلا إذا أدخلت تاريخاً خطأ في الذاكرة المؤقتة للتعديل و التحرير. ويفحص ذلك بعد انتهاء تعديل GET وقبل تعيين القيمة. ينبغي ألا تكون بحاجة مطلقاً لاستخدام هذا المتغير في شيفرة المصدر الخاصة بك.

الكتلة block (يمكن تعيينها)

إنها كتلة الشيفرة التي يستخدمها هدف GET كوسيط لتعديل قيمة المتغير. ويتم إنشاؤها تلقائياً بواسطة وظيفة () _GET_ الداخلية ، أما إذا كنت تستخدم وظيفة () GETNEW فيجب أن تنشئها بنفسك ، كما في المثال التالي:

```
local x := space(20)
setpos(20, 10)
theget := getnew( row() , col() , { | v | IF(pcount() > 0 , x := v , x ) }
```

تسمى كتلة الشيفرة هذه "استرجاع/ تعيين" (أو "get/set") لأنها يمكنها استرجاع قيمة هدف GET أو تعيين قيمة له. (لاحظ أن وظيفتي () FIELDBLOCK و () FIELDWBLOCK يقومان بإنشاء كتل شيفرة بتركيب مماثل تماماً لحقول قاعدة البيانات).

يجب إنشاء كتلة الشيفرة بحيث تقبل قيمة متغير (في هذه الحالة ٥) ، ثم تتحقق من مرور هذه القيمة باستخدام وظيفة (pcount) أو قيمة "الصفري" val = NIL . إذا مُررت قيمة المتغير فيجب تعيين قيمتها لمتغير GET ، وإلا نستخدم قيمة المتغير GET . لذلك يمكن تقيم كتلة الشيفرة بقيم المتغيرات لتعيين قيمة GET ، أو بدون قيم المتغيرات لاسترجاع قيمة GET .

```
EVAL(block)           // retrieve current value of the GET
EVAL(block, 5)        // assign the value 5 th the GET
```

ذاكرة مؤقتة buffer (يمكن تعيينه)

عندما نعدل أمر GET لا نغير متغير GET ولا كتلة الشيفرة ، بل إننا نغير ذاكرة التعديل التي هي سلسلة حرفية ، ولا يهم نوع المتغير الذي تحصل عليه. يتم استهلاك المتغير الفوري "ذاكرة مؤقتة" buffer بواسطة وظيفة (get:setFocus) . وعند الانتهاء من تعديل أمر GET ، تعين وظيفة (assign) محتويات هذه الذاكرة المؤقتة إلى أمر GET الخاص بك. ويتم تحويل الأنواع الضرورية آلياً.

الشحنة cargo (يمكن تعيينه)

يحدد هذا المتغير الفوري من قبل المستخدم. ويمكن استخدامه في عمليات مختلفة ، مثل: رسالة تعرض كلما ظلل هدف GET . ويفضل وضع مصفوفة في الشحنة ، لأنها يمكنها حينئذٍ حمل قيم متعددة وبالتالي تستخدم لأغراض متعددة.

التغيير changed (يمكن تعيينه)

يحتوي هذا المتغير الفوري قيمة منطقية تعتمد على أي تغيير في ذاكرة تعديل GET . وتكون قيمة حقيقية (.T.) فإذا تم تغيير الذاكرة المؤقتة ، وإلا "غير حقيقية" (.F.) .

العمود col (يمكن تعيينه)

يحتوي هذا العمود الفوري رقماً يمثل العمود الذي ستعرض عليه هدف GET في الشاشة. عند إنشاء هدف GET بأمر GET..@ يُستخدم متغير "العمود" <coloumn> لاستهلال هذا المتغير الفوري. ويمكنك تغييره بعد ذلك ، وتوضح العبارة التالية ذلك:

```
local x := 0
local getlist := { }
@ 20, 10 get x           // initially displays GET at @ 20, 10
getlist[1]:col = 50      // GET will be redisplayed at @ 20, 50
read
```

نستخدم في المثال التالي المتغير الفوري get:col لنقل كل هدف من أهداف GET إلى عمود "مدين" Debit أو "دائن" Credit حسب قيمته. لاحظ أنك عندما تختار تحريك أهداف GET على الشاشة فإن عليك إخفاء (مسح) الأهداف من موضعها السابق. نستخدم في هذا المثال وظيفة () SCROLL للقيام بذلك ، ولإعادة حساب وعرض إجمالي كل من عمود Debit و Credit.

```
// filename: GETS10.PRG
function gets10
local a := { 0, 0, 0, 0, 0 }
local getlist := { }
scroll()
@ 8, 30 say "Debits"
@ 8, 50 say "Credits"
@ 10,10 get a[1] valid debitcredit(a)
@ 11,10 get a[2] valid debitcredit(a)
@ 12,10 get a[3] valid debitcredit(a)
@ 13,10 get a[4] valid debitcredit(a)
@ 14,10 get a[5] valid debitcredit(a)
@ 15,30 say replicate(chr(196), 10)
@ 15,50 say replicate(chr(196), 10)
read
return nil
```

```
function debitcredit(altems)
local num_items := len(altems)
local get := getactive()
```

```
local val := get:varGet()
local ntot := 0
local x
//----- blank out GET buffer at current location prior to moving it
scroll(get:row, get:col, get:row, get:col + len(get:buffer), 0)
if val > 0
  get:col = 50
  get:colorDisp("N/BG,+W/BG")
  //----- recalculate total for this column
  for x := 1 to num_items
    if altems[x] > 0
      ntot += altems[x]
    endif
  next
  @ 16, 50 say ntot
elseif val < 0
  get:col = 30
  get:colorDisp("W/R,+GR/R")
  //----- recalculate total for this column
  for x := 1 to num_items
    if altems[x] < 0
      ntot += altems[x]
    endif
  next
  @ 16, 30 say ntot
endif
return .t.
```

طيف الألوان colorSpec (يمكن تعيينه)

إن هذا المتغير هو سلسلة حرفية تشير إلى اللون الذي سيعرض به هدف GET. وصيغته هي : "غير مختار" <unselected> و "مختار" <selected>. فعند إنشاء هدف GET بأمر GET..@ تتسلسل مجموعات الألوان غير المختارة والمختارة وتستخدم لتأسيس طيف الألوان. ولكن يمكنك تغييرها بعد ذلك.

توضح الشيفرة التالية كيفية تغيير لون هدف GET ضمن عبارة "صحيح" VALID باستخدام المتغير الفوري "طيف الألوان" colorSpec. نستخدم فيها وظيفة

"عرض اللون" (colorDisp) التي تغير طيف الألوان وتعيد عرض هدف GET في وقت واحد. تستخدم هذه القاعدة للفت النظر إلى أجزاء معينة من البيانات.

```
local x := 0, getlist := { }
@ 20, 10 get x valid test ( )
read
return nil
```

```
static function test
local get := getactive( )
if get:varGet( ) == 0
    // make GET white on cyan when highlighted ,
    // black on cyan when not
    get:colorDisp("N/BG , +W/BG")
    ret_val := .f.
endif
return nil
```

موضع الفاصلة العشرية decPos

يحتوي هذا المتغير الفوري قيمة رقمية تمثل موضع الفاصلة العشرية في الذاكرة المؤقتة هدف GET. وفي المثال التالي تكون قيمة هذا المتغير الفوري (٤).

```
local x := 0
local gelist := { }
@ 20, 0 get x picture '####.##'
read
```

حالة الخروج exitState

أضيف هذا المتغير للإصدار 5.01 من كليبر. وهو يحتوي قيمة رقمية تبين كيفية الخروج من هدف GET. ويستخدم كثيراً في ملف شيفرة المصدر GETSYS.PRG. اللهم إلا إذا قمت بتطوير قارئة خاصة بك ، وعلى العموم فأنت لست بحاجة لعمل أي شيء مع get:exitState في شيفرتك.

يشتمل الجدول التالي على قائمة بالقيم الممكنة لـ "حالة الخروج" exitState مع "ثوابتها" والمفاتيح التي تستخدم لضبط هذه القيم. لاحظ أن هذه القيم موجودة في ملف الترويسة GETEXIT.CH. فإذا عدلت قيمة "حالة الخروج" exitState في الشيفرة الخاصة بك عليك أن تشير إلى "الثوابت" بدلاً من القيم الرقمية ، لأن الأرقام عرضة للتغيير.

القيمة	ثابت البيان الموجود في ملف الترويسة GETEXIT.CH	المفاتيح المسؤولة
0	GE_NOEXIT	لا يمكن الخروج !
1	GE_UP	مفتاح السهم إلى أعلى
2	GE_DOWN	مفتاح السهم إلى أسفل
3	GE_TOP	مفتاحا [Ctrl]-[Home]
4	GE_BOTTOM	مفتاحا [Ctrl]-[End]
5	GE_ENTER	مفتاح الإدخال [Enter]
6	GE_WRITE	المفاتيح [Ctrl]-[W] ، [PgDn] ، [PgUp]
7	GE_ESCAPE	المفتاح [Esc]
8	GE_WHEN	WHEN هي عبارة تدقيق "غير حقيقي" .F.

استخدام exitState للانتقال من هدف GET إلى آخر

نسخ أولاً برنامج GETSYS.PRG ونسميه باسم آخر. ثم نبحث عن تركيب DO...CASE ، الذي يبدأ في السطر ٤٨٣ (في كليبر 5.01) أو السطر ٤٩٩ (في كليبر 5.2) ، ونضيف هذين السطرين تحت عبارة DO CASE مباشرة:

```
case ( exitState < 0 )
    pos := - exitstate           // use NPOS, not POS , for Clipper 5.2
```

الخطوة التالية هي تجميع نسختك المعدلة من GETSYS.PRG ، ثم جمع الشيفرة التالية واربط الهدفين لإنشاء البرنامج النموذجي.

```
// filename: GETS11.PRG
```

```

#include "inkey.ch"
#include "getexit.ch"

#define PURCHASE_ORDER 11
#define AMOUNT 12

function gets11
set scoreboard off
simpletest()
matrixtest(PURCHASE_ORDER)
matrixtest(AMOUNT)
return nil

function simpletest
local getlist := {}
local x, a := { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
setcursor(3)
scroll()
@ 23, 0 say "Press F1 to jump to a different GET"
set key K_F1 to changeget
for x := 1 to len(a)
    @ x + 2, 0 get a[x] color '+w/r, +w/b'
next
read
return nil

static function changeget
local get := getactive()
local getlist := {}
local newget := 1
@ 24, 0 say "Enter GET to jump to:" get newget picture '##' range 1, 10
read
@ 24, 0
get:exitState := - newget
return nil

static function matrixtest(jumpfield)
local a := { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }
local getlist := {}
local x
local b := { || getactive():exitState := GE_ENTER }
local validblock := { | g | jumptoget(g, getlist) }
local oldf2 := setkey(K_F2, { || getactive():exitstate := - jumpfield } )

//—— redefine all arrow keys to exit GETs
setkey(K_LEFT, b)

```

```

setkey(K_RIGHT, b)
setkey(K_UP, b)
setkey(K_DOWN, b)
scroll()
?? "Press arrow keys to navigate between GETs"
for x := 1 to 13 step 4
    @ x+3, 12 get a[x] color '+w/r, +w/b' when boxme() ;
        valid { |g| jumptoget(g, getlist) }
    @ x+3, 27 get a[x+1] color '+w/r, +w/b' when boxme() ;
        valid { |g| jumptoget(g, getlist) }
    @ x+3, 42 get a[x+2] color '+w/r, +w/b' when boxme() ;
        valid { |g| jumptoget(g, getlist) }
    @ x+3, 57 get a[x+3] color '+w/r, +w/b' when boxme() ;
        valid { |g| jumptoget(g, getlist) }
next
read
//----- turn off left and right arrow keys
setkey(K_F2, oldf2)
setkey(K_LEFT, NIL)
setkey(K_RIGHT, NIL)
setkey(K_UP, NIL)
setkey(K_DOWN, NIL)
return nil

/* draw box around currently active GET for ease of visibility */
static function boxme
local get := getactive()
local nlength
/*
    Pay attention... we activate the GET so that we can determine the
    length of the buffer. Until the GET is active, it has no buffer.
    We need to know this length so that we can draw the box framing
    the GET. The DISPBEGIN() and DISPEND() functions hide the activation
    from the end user.
*/
dispbegin()
get:setFocus()
nlength := len(get:buffer)
get:killFocus()
dispbox(get:row - 1, get:col - 1, get:row + 1, ;
        get:col + nlength, 1, get:colorSpec)
dispend()
return .t.

#define KEY 1
#define JUMP 2

```

```

static function jumptoget(get, getlist)
local key := lastkey()
local maxgets := len(getlist)
static jumps_ := { {K_UP, -4}, ;
                    {K_DOWN, 4}, ;
                    {K_LEFT, -1},;
                    {K_RIGHT, 1} }
local currgetno := get:subscript[1]
local ele := ascan(jumps_, { | info | info[KEY] == key })
if ele > 0
do case
case currgetno + jumps_[ele][JUMP] < 1
get:exitState := - ( currgetno + jumps_[ele][JUMP] + maxgets )
case currgetno + jumps_[ele][JUMP] > maxgets
get:exitState := - ( currgetno + jumps_[ele][JUMP] - maxgets )
otherwise
get:exitState := - ( currgetno + jumps_[ele][JUMP] )
endcase
endif
/* remove the box surrounding this GET */
dispbox(get:row - 1, get:col - 1, get:row + 1, get:col + len(get:buffer), ;
        space(8))
return .t.

```

يلاحظ في المثال أعلاه أن جميع مفاتيح الأسهم قد أعيد تحديدها بواسطة وظيفة "ضبط المفاتيح" (SETKEY) لضبط "حالة الخروج" exitState من هدف GET الحالي. كما يلاحظ أن لكل هدف من أهداف GET عبارة "صحيح" VALID المرتبطة به والتي تستدعي (JumpToGet). إن عبارات "صحيح" VALID هذه مكتوبة بصيغة كتل شيفرة. ولهذا ، يمكننا تمرير الإشارة إلى المصفوفة Getlist الحالية إلى الوظيفة الصحيحة. ستجد مع الوقت أن هذا ضروري للغاية ، كما تعمقت في العمل مع المصفوفات المحلية getlist ، وهذا بالفعل سيكون أفضل من المصفوفة العامة Getlist.

ستلاحظ أيضاً أن عبارات "صحيح" VALID تقبل المتغير G. وهذا يمثل هدف GET العامل حالياً. والذي سيمرر تلقائياً بواسطة الوظيفة (GetPostValidate) في ملف شيفرة المصدر GETSYS.SYS.

تفحص وظيفة () JumpToGet آخر ضغطة مفتاح وتمسح مصفوفة معلومات المفاتيح. تحتوي هذه المصفوفة أربعة عناصر: عنصر لكل مفتاح سهم. فإذا كان المفتاح المضغوط موجوداً في المصفوفة ، يعدل المتغير الفوري "حالة الخروج" exitState الخاص بهدف GET الحالي وفق ذلك ، وإلا تضبط قيمة "حالة الخروج" exitState إلى الصفر (GE_NOEXIT) بحيث يمكنك متابعة تعديل هدف GET ذاك. لاحظ كيف تستخدم وظيفة () JumpToGet المتغير الفوري "الرمز السفلي" subscript لتحديد هدف GET الذي تعمل عليه حالياً.

لكل هدف GET عبارة WHEN التي تستدعي وظيفة () BoxMe ، التي تشغل بدورها هدف GET بحيث يمكننا تحديد طول الذاكرة المؤقتة (ذكرنا آنفاً أنه لا يكون لهدف GET ذاكرة مؤقتة buffer مالم يشغل للتعديل) ويجب معرفة هذا الطول لتحديد إحداثيات المربع. تخفي وظيفة "بداية العرض" () DISPBEGIN و "نهاية العرض" () DISPEND خطوة تشغيل الهدف عن مستخدم البرنامج. وبالتالي يمسح المربع في أسفل وظيفة () JumpToGet.

لاحظ أن وظيفة () MatrixTest تقبل متغيراً رقمياً ، يمكن استخدامه لضبط مفتاح الاستخدام السريع hotKey (في هذه الحالة [F2]) الذي ينقلنا إلى حقل معين.

مظلل hasFocus

يحتوي هذا المتغير الفوري قيمة منطقية "حقيقية" (T.) عندما يظل هدف GET (أي له تركيز إدخال بيانات) و "غير حقيقية" (F.) عندما لا يكون مظللاً. يعين تركيز الإدخال بواسطة وظيفة "ضبط التركيز" () setFocus الذي يستدعي من ضمن وظيفة () GetReader. يبين المثال التالي كيف يغير المتغير الفوري "مظلل" () et:hasFocus القيمة.


```

local x := 0 , getlist := { }
@ 20, 10 get x
? getlist[1]:hasFocus // .F.
inkey(0)
getlist[1]:setFocus( )
? getlist[1]:hasFocus // .T.

```

الاسم name (يمكن تعيينه)

هذه سلسلة حرفية تحتوي اسم المتغير GET. يتم استهلاك هذا المتغير تلقائياً عند إنشاء أهداف GET بأمر GET..@ ، ويجب تمريره كمتغير إلى وظيفة GETNEW(). يستخدم هذا المتغير "name" لأغراض التعريف. فيعين لوظيفة READVAR() عند تظليل هدف GET.

إن تغيير اسم متغير GET لن يؤثر في تحديد المتغير المعدل بل تحدده كتلة الشيفرة المجهزة سابقاً والمربطة بهدف GET. ففي المثال التالي ، مع أنه تم تغيير المتغير الفوري "name" إلى (Y) مازال المتغير الذي نعالجه هو (X) ، اضغط مفتاح [F1] في GET لفحص القيمة الحالية للوظيفة READVAR().

```

// filename: GETS12.PRG
#include "inkey.ch"

function gets12
local getlist := {}
local x := 0, y := 5
set key K_F1 to testreadvar
scroll()
@ 10,10 get x
getlist[1]:name = 'y'
read
? "x = ", x // whatever you changed it to
? "y = ", y // still 5
inkey(0)
return nil

static function testreadvar
@ 0,0 say "READVAR() = " + readvar() // Y, not X
return nil

```

القيمة الأصلية original

المتغير الفوري هذا هو سلسلة حرفية تحتوي نسخة من القيمة الأصلية للذاكرة GET المؤقتة. ويمكن عند الضرورة إعادة ضبط قيمة الذاكرة المؤقتة إلى قيمتها الأصلية بواسطة وظيفة () undo . ويظهر ذلك في وظيفة () GetApplyKey عندما تضغط مفتاح [Esc] للخروج من هدف GET.

```
/* excerpted from GetApplyKey( ) */
case ( key == K_ESC )
  if ( set(_SET_ESCAPE) )
    get:undo( )
    get:exitState := GE_ESCAPE
  end
```

الصورة picture (يمكن تعيينه)

هي سلسلة حرفية تحدد عبارة "الصورة" PICTURE التي ستستخدم لعرض ذاكرة GET المؤقتة. وسيعين عندما تعين عبارة "الصورة" PICTURE بأمر GET..@ أو إذا مرت متغيراً ملائماً إلى وظيفة () GETNEW. وإذا لم يعين ، سيكون له قيمة "الصفري" NIL. ويمكنك دائماً تغييرها بعد ذلك. لاحظ أنه تم عرض هدف GET ثم غيرت PICTURE أثناء التشغيل on-the-fly ، فيجب عليك حذف أجزاء لـ: GET الأصلية.

بين الشيفرة التالية مثلاً لتغيير عبارة الصورة أثناء التشغيل. يمكن أن يكون الرقم، رقم البطاقة الشخصية (رقم التأمين الاجتماعي) أو رقم بطاقة العمل (رقم الإقامة).

```
// filename: GETS13.PRG
function gets13
local id := space(9)
local personal := .t.
local getlist := {}
```

```

scroll()
@ 1,1 say "Personal?" get personal picture "Y"
@ 2,1 say "ID number" get id when changepic(personal) picture '@R 999-99-9999'
read
return nil

static function changepic(val)
if ! val
  getactive():picture := "@R 99-99999999"
endif
return .t.

```

موضع المؤشر POS (يمكن تعيينه)

يحتوي هذا المتغير الفوري قيمة رقمية تمثل الموضع الحالي للمؤشر في ذاكرة التحرير المؤقتة. ويمكنك المثال التالي من اختبار ذلك بنفسك بواسطة ضغط مفتاح [F1] حينما تكون في هدف GET.

```

// filename: GETS14.PRG
#include "inkey.ch"

function gets14
local getlist := {}
local x := "Press F1 to see where you are"
scroll()
set key K_F1 to testpos
@ 20,0 get x
read
return nil

static function testpos
@ 0,0 say "Current cursor position: " + str(getactive():pos)
return nil

```

المثال التالي أوضح قليلاً ، فهو يمكنك من إدراج اسم في هدف GET في الموضع الحالي للمؤشر. حرك المؤشر الى موضع الفاصلة ثم اضغط مفتاح [F2] لعرض قائمة بالاسماء. وبعد أن تختار اسماً من هذه القائمة ، نعدل المتغير الفوري "موضع المؤشر" pos مما يضمن احتفاظ المؤشر بموضعه الأصلي في ذاكرة GET المؤقتة قبل الإدراج.

```
// filename: GETS14A.PRG
#include "inkey.ch"

function gets14a
local m_var := "Mr. , Director of Communications", getlist := {}
set key K_F2 to showfields
scroll()
@ 1, 20 get m_var
@ 2, 22 say "Press F2 to select from list of names"
read
? m_var
return nil

static procedure showfields
local get := getactive()
local position := get:pos, val := get:varGet()
local names := { "Andersen", "Creagh", "Jones", "Lief", "Worthen" }
local ele := 0, xx, oldscm := savescreen(9, 35, 15, 44)
@ 9, 35 to 15, 44
//---- don't allow Esc, which would cause array access error
do while ele == 0
    ele := achoice(10, 36, 14, 43, names)
enddo
//---- drop selected name into GET at current position
get:varPut(substr(val, 1, position - 1) + names[ele] + ;
            substr(val, position))
//---- move cursor to original location in GET buffer,
//---- based on length of the name we just inserted
get:pos := position + len(names[ele])
restscreen(9, 35, 15, 44, oldscm)
return
```

كتلة لاحقة postBlock (يمكن تعيينه)

يحتوي هذا المتغير الفوري كتلة شيفرة تستخدم لتدقيق قيمة مدخلة لهدف GET. فإذا عينت عبارة "صحيح" VALID بأمر GET..@ ستحول إلى كتلة شيفرة وتخزن في المتغير الفوري "كتلة لاحقة" postBlock.

وإذا لم تستخدم عبارة "صحيح" VALID فسيحوي postBlock قيمة "الصفء" NIL. بعد الخروج من هدف GET ، تفحص وظيفة (GetPostValidate)

المتغير "كتلة لاحقة" postBlock للتأكد من تعيين عبارة "التدقيق اللاحق" ، وفي هذه الحالة سيتم تقييم كتلة الشيفرة (بتمرير هدف GET الحالي كمتغير).

يمكنك تغيير "كتلة لاحقة" postBlock بعد ذلك إذا أردت. كما يمكنك أيضاً تغييرها داخل وظائف VALID الخاصة بك.

فكرة مفيدة

من المعروف أن نظام GET في كليبر 5.01 يمرر هدف GET العامل حالياً كمتغير إلى وظيفة "صحيح" VALID الخاص بك. لذلك يجب كتابة وظيفة VALID بصيغة كتلة شيفرة بحيث يمكنك ضبطه ليقبل المتغير اللازم. وسنوضح ذلك في الفقرة التالية.

كتلة سابقة preBlock (يمكن تعيينه)

يشبه هذا المتغير الفوري ما قبله باستثناء أنه يتم تقييمه قبل أن ندخل هدف GET ، وليس بعد أن نخرج منه ، فإذا عينت عبارة WHEN بأمر GET..@ فستحول إلى كتلة شيفرة وتخزن في المتغير الفوري "كتلة سابقة" preBlock. وإذا لم تستخدم عبارة WHEN فسيحتوي preBlock قيمة "الصفري" NIL. وقبل أن ندخل هدف GET ، تفحص وظيفة () GetPreValidate المتغير "كتلة سابقة" preBlock للتأكد من وجود عبارة "تدقيق سابقة" ، وفي هذه الحالة سيتم تقييم كتلة الشيفرة ويحدد إذا كان يمكن دخول هدف GET ، أم لا (بتمرير هدف GET الحالي كمتغير). يمكن تعديل هذا المتغير كما نريد.

كما هي الحال مع المتغير "كتلة لاحقة" postBlock ، يمكن الاستفادة من نظام GET في كليبر 5.01 ، بتمرير هدف GET العامل حالياً إلى وظيفة WHEN الخاص بك.

تستدعي عبارة WHEN و VALID ، في المثال التالي ، وظيفة "حركة" MoveIt() التي تقبل هدف GET العامل كمتغير ، وتغيير موضع هدف GET بينما تقوم بتعديله وتحريره ، مما يحدد للوظيفة اتجاه العمل. لاحظ أن كلتا العبارتين WHEN و VALID كتبتا بصيغة كتل شيفرة.

```
// filename: GETS15.PRG
function gets15
local a := { 1, 2, 3, 4 }, y
local getlist := {}
scroll()
for y := 1 to 4
  @ y, 1 get a[y] when { | g | moveit(g, .t.) } ;
  valid { | g | moveit(g, .f.) }
next
read
return nil

static function moveit(get, lmoveit)
static nrow, ncol
local oldcolor
if lmoveit
  //----- save current position of GET
  nrow := get:row
  ncol := get:col
  oldcolor := get:colorSpec
  //----- blank out GET at current position... must play with color
  //----- instead of using SCROLL(), because buffer instance variable
  //----- will not yet have been initialized
  get:colorDisp("N/N,N/N")
  get:row := 12
  get:col := 20
  get:colorDisp(oldcolor)
elseif nrow != NIL .and. ncol != NIL
  scroll(get:row, get:col, get:row, get:col + len(get:buffer), 0)
  get:row := nrow
  get:col := ncol
endif
return .t.
```

القارئ reader (يمكن تعيينه)

أضيف هذا المتغير الفوري للإصدار 5.01 من كليبر. ويمكن استخدامه لقراءة بعض أهداف GET. فإذا احتوى `get:reader` كتلة شيفرة ، ستقوم وظيفة `READMODAL()` تلك الكتلة لكي تقرأ هدف GET (يمرر هدف GET كمتغير للكتلة). يمكن للكتلة بعد ذلك أن تستدعي أية وظيفة مطلوبة لتوفر لها إمكانية تعديل وتحرير هدف GET. أما إذا لم تحتو `get:reader` كتلة شيفرة ، فتستخدم وظيفة `READMODAL()` إجراء قراءة افتراضية لهدف GET.

يتيح هذا المتغير الفوري وجود إجراء قراءة خاص لهدف GET معينة دون أن يتطلب ذلك تعديلك لوظيفة `READMODAL()` القياسية.

المتغير الفوري "مرفوض" rejected

يحتوي هذا المتغير الفوري قيمة منطقية تشير إلى آخر رمز عيّن بواسطة وظيفة "إدراج" `insert()` أو وظيفة "كتابة فوقية" `overstrike()` ، قد انتقلت فعلاً إلى ذاكرة التعديل المؤقتة `buffer`. وستحتوي قيمة "غير حقيقي" (F.) إذا وضع الرمز في الذاكرة المؤقتة ، وإلا فستحتوي قيمة "حقيقي" (T.) إذا رفض. وإن إدخال أية رسالة نص لاحق سيسبب في إعادة ضبط هذا المتغير إلى الوضع السابق.

المتغير الفوري "الصف" row (يمكن تعيينه)

يحتوي هذا المتغير الفوري رقماً يمثل الصف الذي سيعرض عليه هدف GET في الشاشة عند إنشاء هدف GET بأمر `GET.@` سيستخدم متغير "الصف" `<row>` لتأسيس هذا المتغير الفوري. ويمكنك تغييره متى شئت.

function main

```
local x := 0
local getlist := { }
@ 20, 10 get x           // initially displays GET at @ 20, 10
getlist[1]:row = 11      // GET will be redisplayed at @ 11, 10
read
return nil
```

المتغير الفوري "الرمز الفرعي" subscript

لقد أضيف هذا المتغير الفوري للإصدار 5.01 من كليبر ، لمعالجة المشكلة التي دامت طويلاً مع كليبر ، وهي عدم القدرة على معرفة عنصر المصفوفة التي تعمل عليها الآن. وهذا يسبب مشكلة عويصة ، عندما تحاول ربط شاشات المساعدة لكل GET.

يحدد هذا المتغير الفوري عنصر المصفوفة المتضمن لهدف GET. فإذا كان هدف GET عنصراً في مصفوفة فسيحتوي هذا المتغير الفوري مصفوفة من عنصر واحد لكل رمز فرعي. وإذا لم يكن هدف GET عنصر مصفوفة فسيحتوي هذا المتغير الفوري قيمة "الصفري" NIL.

تحذير

يمكنك كليبر 5.2 من تعيين قيمة "الرمز الفرعي" get:subscript ، وفي هذه الحالة تأكد من تعيينه مصفوفة تحتوي قيماً رقمية. أي انحراف سينتج عنه خطأ في وقت التنفيذ ("argument error: LEN").

```
// filename: GETS16.PRG
function gets16
local x := 0, y := { 1, 2, 3 }, z := { { 1, 2 }, { 3, 4 } }
local getlist := {}
scroll()
@ 10, 10 get x
@ 11, 10 get y[3]
@ 12, 10 get z[2, 1]
? getlist[1]:subscript    // NIL
```

```
? getlist[2]:subscript[1] // 3
? getlist[3]:subscript[1] // 2
? getlist[3]:subscript[2] // 1
inkey(0)
read
return nil
```

نستخدم في المثال التالي المتغير الفوري "رمز فرعي" subscript لتحديد هدف GET الذي نعمل عليه ، ونعرض رسالة مساعدة مناسبة. ومع أن رسائل المساعدة هذه مشفرة في البرنامج ، يمكن تطبيق القاعدة في نظام المساعدة الخاصة بالسياق -context-specific.

```
// filename: GETS17.PRG
#include "inkey.ch"

function gets17
local a := { padr("John",20), padr("Doe", 20), padr("123 Main Street", 30), ;
             padr("Anytown", 25), "MD", padr("21157", 10) }
local getlist := {}
set key K_F1 to helpme
scroll()
@ 10,0 get a[1]
@ 11,0 get a[2]
@ 12,0 get a[3]
@ 13,0 get a[4]
@ 14,0 get a[5]
@ 15,0 get a[6]
read
return nil

static function helpme
local get := getactive()
static helpmsg := { "first name", "last name", "address", ;
                    "city", "state", "zip"}
if ! empty(get:subscript)
    @ maxrow(), 0 say "This is the " + helpmsg[get:subscript[1]] + ;
    "... press a key"
    inkey(0)
    scroll(maxrow(), 0)
endif
return nil
```

المتغير الفوري "النوع" type

إن هذا المتغير الفوري هو سلسلة حرفية تحدد نوع متغير GET. إن كليبر يستلزم معرفة نوع البيانات الذي ستحول إليه معلومات الذاكرة المؤقتة get:buffer. وتفيد معرفة نوع البيانات لإجراء عمليات على المفاتيح الخاصة بأنواع معينة من البيانات. فمثلاً: إذا كنت تريد إعادة تعريف مفتاحي (+) و (-) لزيادة و إنقاص متغير GET (وهذا ماسنعمله قريباً) وهذا يلائم التواريخ والأرقام فقط. كما ينبغي عليك القيام بفحص get:type قبل محاولة زيادة السلسلة الحرفية.

يؤسس المثال التالي أربعة أهداف GET ويمكنك من فحص كل واحد خاص بكل نوع بضغط مفتاح [F1]. تستدعي الوظيفة () TESTTYPE وظيفة كليبر () GETACTIVE التي تعيد هدف GET العامل حالياً.

```
// filename: GETS18.PRG
#include "inkey.ch"

function gets18
local getlist := {}
local w := "Press F1 to test type of each GET"
local x := 0, y := .t., z := date()
set key K_F1 to testtype
scroll()
@ 11, 0 get w
@ 12, 0 get x
@ 13, 0 get y
@ 14, 0 get z
read
return nil

static function testtype(p, l, v)
local type := { "Character", "Numeric", "Date", "Logical" } ;
               [at(getactive():type, "CNDL")]
@ 0,0 say "Current variable is " + padr(type, 9)
return nil
```


متغير "الخروج من الذاكرة" typeout

يحتوي هذا المتغير الفوري قيمة منطقية "حقيقية" (T.) فإذا حاولت نقل المؤشر خارج ذاكرة التعديل أو إذا لم يوجد مواضع قابلة للتعديل في ذاكرة التعديل. وسيعاد ضبط قيمته بواسطة أي من وظائف تحريك المؤشر في ذاكرة GET المؤقتة.

تشغيل المتغيرات الفورية

لقد جعل كليبر Summer'87 المبرمجين يلتفون حول كل أنواع وطرق البرمجة المؤلة بحثنا عن طريقة تمكنهم من التحكم في أوامر GET. فيما يلي نموذج لما يطلبه مبرمجو كليبر من شاشة إدخال البيانات.

المسألة

توجد ثلاثة أهداف GET في العمود. يمكن أن يفترض كل منها قيمة بين "الصفري" و "٣". وتسبب كل واحدة من هذه القيم الإجراءات التالية:

- الصفري: تقدم إلى هدف GET التالي.
 - (١) : يتيح للمستخدم إدخال رقم في العمود الثاني ، باستخدام عبارة "#####.###" الخاصة بالصورة.
 - (٢) : يتيح للمستخدم إدخال رقم في العمود الثاني ، باستخدام عبارة "###.###" الخاصة بالصورة PICTURE.
 - (٣) : يتيح للمستخدم إدخال رقمين في العمودين الثاني والثالث ، باستخدام عبارة "###.###" الخاصة بالصورة PICTURE.
- القراءة المتداخلة ليست خياراً عملياً ، وذلك لأنها تستلزم من المستخدم أن يكون قادراً على التحرك بين كل أوامر GET ، بما في ذلك الذي في العمود الثاني والثالث (إذا كان هناك شيء).

وكما قد يخطر ببالك ، فإن هذا كان كابوساً مزعجاً في كليبر Summer'87 ، حتى مع استخدام المنتجات الإضافية لتطوير كليبر من الشركات المساندة لكليبر. فهي تأخذ مايساوي ٣٠٠ سطر ، من السطور المتداخلة حتى يمكن للشيفرة تأدية عملها ،

ومع ذلك فهي تعمل بشق الأنفس. ولحسن الحظ ، فإن كليبر حل هذا الإشكال مع وجود المنهج الجديد ، هدف GET ومصفوفة GETLIST والتي جعلت من هذا النوع شيئاً غاية في السهولة واليسر.

الحل

استخدم مصفوفات بدلاً من متغيرات الذاكرة

أولاً ، لقد أنشأنا مصفوفةً لتضمين العناصر فيها بدلاً من متغيرات الذاكرة ، والسبب هو أن العناصر تكون مرتبطة منطقياً أكثر منها في متغيرات الذاكرة ، فإن استخدام الربط المنطقي هام لحل هذه المسألة.

تحتوي هذه المصفوفة ثلاثة عناصر ، كل منها مصفوفة بذاتها تحتوي ثلاثة عناصر أخرى. فيكون تصميم الشاشة على النحو التالي:

a[1, 1]	a[1, 2]	a[1, 3]
a[2, 1]	a[2, 2]	a[2, 3]
a[3, 2]	a[3, 2]	a[3, 3]

أيضاً يعتبر استخدام مصفوفة GETLIST هاماً لحل هذه المسألة ، لأن هدي GET في العمودين الثاني والثالث يعتمد على هدف GET في العمود الأول. ولقد استفدنا كثيراً من إمكانية استعراض مصفوفة GETLIST لمعرفة أهداف GET الأخرى (وتعديلها).

أهداف GET غير المرئية

الخطوة التالية هي تحديد كيفية تنفيذ أهداف GET الاختيارية في العمودين الثاني والثالث. يحتوي هدف GET كافة المتغيرات الفورية التي نستخدمها للتحكم في الانتقال بين كافة أهداف GET لإجراء "القراءة المتداخلة". ورد معنا أن "طيف الألوان" `get:colorSpec` يمكننا من تغيير لون هدف GET حسبما نريد وهذا يمكننا من جعل العمودين الثاني والثالث غير مرئيين (أو مرئيين عند اللزوم).

إننا نعرف بالضبط أين يظهر هدف GET على الشاشة بواسطة المتغيرين الفوريين "الصف" `get:row` و "العمود" `get:col`. وهذا يساعدنا في هدف GET الحالي إن أردنا ذلك.

تعديل الصورة Picture

لدينا إمكانية تغيير عبارة PICTURE أثناء التشغيل بواسطة تعديل المتغير الفوري "الصورة" `get:picture`. وهذا فضل من الله ، ذلك لأن لأمر GET في العمود الثاني احتمالين للصورة التي تظهر عليها بياناته ("`#####.###`" أو "`###.###`").

إنقاص عبارة WHEN

لم يكن بالإمكان إجراء أي من العمليات السابقة لولا عبارة WHEN. وبما أن أهداف GET في العمودين الثاني والثالث تعتمد بشكل كامل على ما يحتويه العمود الأول الموافق ، فإنه يجب تعديلها قبل أن يتمكن المستخدم من الدخول إليها. وإن لم تكن هناك حاجة لأهداف GET في العمودين الثاني والثالث ، فتمنع وظيفة WHEN المستخدم

من الدخول إليها ، وبما أنها تعرض بشكل غير مرئي (أسود على خلفية سوداء) فلا يمكن للمستخدم إدراك عدم وجودها.

بالنسبة لكافة أهداف GET في العمودين الثاني والثالث ، تمرر مصفوفة getlist كمتغير لوظيفة WHEN ، التي تسمح هذه المصفوفة لتحديد موضع هدف GET الحالي. ويبحث النظام عن أول هدف GET في مصفوفة Getlist التي توافق المتغيرين الفوريين name و subscript المرتبطين بهدف GET الحالي.

بعد تحديد موضع هدف GET الحالي/نشاهد محتويات هدف GET في العمود الأول:

```
local value := getlist[ele - 1]:varGet( )
```

استخدمنا هذه العبارة الخاصة عندما كنا في العمود الثاني. وكان ELE يمثل الموضع الحالي للمؤشر ، لذلك كان علينا إنقاص رقم (١) لمشاهدة هدف GET في العمود الأول.

عدلت صورة ولون كل هدف من أهداف GET في العمودين الثاني والثالث اعتماداً على محتويات العمود الأول. لاحظ أننا استخدمنا وظيفة "عرض الألوان" colorDisp() ، المساوٍ لتغيير المتغير الفوري "طيف الألوان" colorSpec وإصدار وظيفة "عرض" display() (التي تعيد عرض هدف GET باللون الجديد).

```
// filename: GETS19.PRG
#define INVISIBLE 'B/B,B/B'
#define VISIBLE '+W/R,+W/BG'

function gets19
local x
local a := { { 0, 0, 0 }, ;
             { 0, 0, 0 }, ;
             { 0, 0, 0 } }
local getlist := {}
setcolor('+gr/b')
scroll()
@ 8, 0 say "Enter values between 1 and 3 to activate different GETs"
@ 10,10 get a[1, 1] picture '#' range 0, 3 color VISIBLE
@ 10,20 get a[1, 2] picture '###.###' color INVISIBLE when checkitem1(getlist)
@ 10,30 get a[1, 3] color INVISIBLE picture '###.###' when checkitem2(getlist)
```



```
@ 11,10 get a[2, 1] picture '#' range 0, 3 color VISIBLE
@ 11,20 get a[2, 2] picture '##.##' color INVISIBLE when checkitem1(getlist)
@ 11,30 get a[2, 3] color INVISIBLE picture '##.##' when checkitem2(getlist)
@ 12,10 get a[3, 1] picture '#' range 0, 3 color VISIBLE
@ 12,20 get a[3, 2] picture '##.##' color INVISIBLE when checkitem1(getlist)
@ 12,30 get a[3, 3] color INVISIBLE picture '##.##' when checkitem2(getlist)
read
return nil
```

```
static function checkitem1(getlist)
local ret_val := .t.
local get := getactive()
//----- the following line scans the GETLIST array to determine our
//----- current position. Once we have this, we can easily determine
//----- the next and previous GETs
local ele := ascan(getlist, ;
    { | g | g:name == get:name .and.
      g:subscript[1] == get:subscript[1] .and.
      g:subscript[2] == get:subscript[2] } )
local value := getlist[ele - 1]:varGet()
//----- if we are not using option 3, we must clear the 3rd column
if value != 3
    getlist[ele + 1]:varPut(0)
    getlist[ele + 1]:colorDisp(INVISIBLE)
else
    //----- make GET in third column visible if it isn't already
    if getlist[ele + 1]:colorSpec == INVISIBLE
        getlist[ele + 1]:colorDisp(VISIBLE)
    endif
endif
do case
    case value == 1
        if get:picture != '#####.##'
            get:varPut(0)
        endif
        get:picture := '#####.##'
    case value > 0
        if get:picture != '##.##'
            get:varPut(0)
        endif
        scroll(get:row, get:col, get:row, get:col + len(get:picture), 0)
        get:picture := '##.##'
    otherwise
        ret_val := .f.
endcase
if ret_val
    get:colorDisp(VISIBLE)
```

```
else
  get:colorDisp(INVISIBLE)
endif
return ret_val
```

```
static function checkitem2(getlist)
local ret_val := .t.
local get := getactive()
local ele := ascan(getlist, ;
  { | g | g:name == get:name .and.
    g:subscript[1] == get:subscript[1] .and.
    g:subscript[2] == get:subscript[2] } )
//--- make this GET visible and editable only if we are using option 3
if getlist[ele - 2]:varGet() == 3
  get:colorDisp(VISIBLE)
else
  get:colorDisp(INVISIBLE)
  ret_val := .f.
endif
return ret_val
```

أهداف GET مشغلة بواسطة البيانات Data-Driven

إن مفهوم التشغيل بواسطة البيانات بسيط ، وهو: اظهار "فصل" البيانات من ملفك القابل للتنفيذ بحيث يمكن تغيير البيانات بسرعة دون الحاجة لإعادة إنشاء البرنامج التنفيذي بأكمله ، وتتماز هذه الطريقة بالمرونة وبالإمكانية المحسنة للمتابعة والتحديث. ومع ذلك فيها شيان: (أ) تكون برامج التشغيل بالبيانات عادة أبطأ بسبب الوقت اللازم لقراءة البيانات من الأسطوانة الصلب ، ويمكن التغلب على هذه السينة بتنفيذ البرنامج التطبيقي على جهاز سريع (مثل: 386/sx أو أفضل) ، (ب) إن وجود البيانات في ملفات (.DBF) أسوأ من الناحية الأمنية من وجودها في ملف (.EXE).

إذا استخدمنا هذه الطريقة ، سنحتاج إلى مكان نحفظ فيه المعلومات الخاصة بشاشات أهداف GET. لذلك سننشئ ملفاً سنسميه GETINFO.DBF ، وسيكون له البنية التالية:

التوضيح	عشرية	العرض	النوع	اسم الحقل
السطر الذي ستعرض عليه التوجيه	0	2	N	ROW
العمود الذي ستعرض عليه التوجيه	0	2	N	COL
السطر الذي ستعرض عليه GET	0	2	N	GETROW
العمود الذي ستعرض عليه GET	0	2	N	GETCOL
توجيه (@..SAY) المرتبطة بـ: GET		50	C	PROMPT
اسم حقل قاعدة البيانات لـ: GET		10	C	FIELDNAME
فقرة الصورة		25	C	PICTURE
اللون الذي ستعرض به التوجيه		6	C	SAYCOLOR
اللون الذي ستعرض به GET		12	C	GETCOLOR
فقرة التأكد من الصحة VALID		35	C	VALID
فقرة WHEN		35	C	WHEN
الصيغة		50	C	FORMULA

إن أقل معلومات نحتاجها لتجهيز هدف GET هي "الصف" GETROW ، و "العمود" GETCOL ، و "اسم الحقل" FIELDNAME.

فإذا اخترت أن تستخدم حقلي VALID و WHEN فيمكنك تعيين عبارة كليبر ذاتها أو بصيغة كتلة شيفرة. ومعنى آخر ، فإن كلا الصيغتين التاليتين صحيحة:

```
notempty( )
{ | g | : empty(g:varGet( ) ) }
```

إذا كتبت شيفرة مشابهة للمثال الأول ، فستحول تلقائياً إلى كتلة شيفرة. ولكن المثال الثاني أفضل لأنه أعم. وبما أن كافة أهداف GET موجودة بمصفوفة aHold فعليك أن تعتد كتابة كتل شيفرة عامة تقبل هدف GET كمتغير وتعمل عليها وفقاً لذلك ، عادة بواسطة وظيفة () varGet.

ستخزن مصفوفة aHold أيضاً مواضع الحقل بالترتيب في هيكل قاعدة البيانات لتسهيل عمليات الاستبدال بعد انتهائنا من القراءة READ.

وإذا استلزم الأمر تغييراً ، يمكنك تغيير المعلومات في ملف GETINFO.DBF بدلاً من إعادة تجميع البرنامج.

الصيغة Formulae

يمكننا حقل "الصيغة" FORMULA في ملف GETINFO.DBF من تشكيل أية صيغة يحددها المستخدم. يستخدم في المثال التالي لعرض رقم السجل الحالي:

```
{ | | 'record #' + ltrim(str(recno( ) ) ) }
```

يجب أن تستخدم PROMPT للنص البسيط ، أو "الصيغة" FORMULA للبيانات القابلة للتغيير (مثل رقم السجل). وإذا وجدت "صيغة" Formula ، فستقيم ثم تعرض

محتوياتها في الموضع الذي يحدده حقلاً "الصف" ROW و "العمود" COL. كما سيستخدم أي لون محدد بواسطة SAYCOLOR لعرض نتيجة "الصيغة" Formula.

تحذير

إذا أشرت إلى أية وظيفة في أحد الحقول VALID و WHEN أو FORMULA فعليك أن تتأكد من وجود هذه الوظائف في برنامجك التطبيقي ، ومن أنها غير محددة بأنها "ساكنة" STATIC لأن الوظائف "الساكنة" STATIC غير مرئية بالنسبة لكتل الشيفرة المجمعة أثناء التشغيل.

قيم محلية مستقلة Detached Locals

يوجد في شيفرة المصدر وظيفة اسمها "تشكيل كتلة" MakeBlock تستخدم فيها "قيم محلية مستقلة" لتشفير الإشارة إلى العنصر الأعلى بالترتيب (أحدث عنصر أضيف) في مصفوفة aHold في البرنامج. وهذا ضروري لأننا بدونه سنشير إلى متغير في كتلة الشيفرة ، ولا تُحل الإشارات إلى المتغيرات ما لم يتم فعلياً تقويم الكتلة.

في كليبر 5.01 كلما إعادة وظيفة ما ، كتلة شيفرة معينة ، فإن القيم المحلية LOCALs في كتلة الشيفرة تبقى عاملة مادامت كتلة الشيفرة باقية.

الكبسلة Encapsulation

لاحظ أن جميع المعلومات الخاصة بصيغة Prompt و Formulae (أي "الصف" row و "العمود" column و "اللون" color و "النص" text و كتلة الصيغة) قد خزنت في مصفوفة موجودة في المتغير الفوري get:cargo. وهذا يربط المعلومات بهدف GET ويمكننا من عرضها لاحقاً إذا لزم الأمر.

لا يتيح لنا كليبر 5.2 إنشاء فئاتنا الخاصة بالأهداف ولا أن ننسخ (وهي ما يطلق عليها: الوراثة) من فئات الأهداف الأربعة ، ولكن إذا كنت تمتلك برنامج Class(y) أو SuperClass ، فيمكنك بسهولة استخدام فئة هدف GET كليبر لإنشاء هدف GET جديد بالمتغيرات الفورية اللازمة (مثل: PromptRow و PromptCol و PromptText و PromptColor و الصيغة). وهذا أفضل من استخدام get:cargo لأنك يمكنك أيضاً كتابة "توجيه العرض" () displayPrompt لعرض نص التوجيه أو الصيغة. فإذا كنت تمتلك رزمة Class(y) أو SuperClass ، فهنا فرصتك لاختبار إعادة كتابة المثال التالي باستخدام الوراثة الفعلية.

```
// filename: GETS21.PRG
#include "inkey.ch"

#define TEST          // to compile test stub

#ifdef TEST          // begin test stub

function gets21
local x
local y
local aTestdata
if ! file('customer.dbf')
    dbcreate('customer', { ;
        { "NAME", "C", 25, 0 }, ;
        { "ADDR", "C", 35, 0 }, ;
        { "CITY", "C", 20, 0 }, ;
        { "STATE", "C", 2, 0 }, ;
        { "ZIP", "C", 10, 0 } ;
    })
    use customer new
    //----- create a blank record for testing
    dbappend()
    dbclosearea()
endif
if ! file('getinfo.dbf')

    aTestdata := { ;
        { 0, 28,,, "Data-Driven Entry Screen", , , '+w/r', , , },
        { 0, 65,,,,,, "{ || 'record #' + ltrim(str(recno())) }" },
        { 1, 26,,, "Copyright (c) 1993 Greg Lief", , , '+w/rb', , , },
        { 3, 1,,, "Customer Name:", "name", "@!", " ", , , },
```

```

        { 4, 1,,, "Address:", "addr", "@!", "", "", ., }, ;
        { 5, 1,,, "City:", "city", "@!", "", "", ., }, ;
        { 6, 1,,, "State:", "state", "@!", "", "", "notEmpty()", ., }, ;
        { 6, 13,,, "Zip/P.C.:", "zip", "#####-####", "", "", ;
            "{ | g | ! empty(g:varGet()) }", ., }, ;
    }

dbcreate('getinfo', { ;
    { "ROW", "N", 2, 0 }, ;
    { "COL", "N", 2, 0 }, ;
    { "GETROW", "N", 2, 0 }, ;
    { "GETCOL", "N", 2, 0 }, ;
    { "PROMPT", "C", 50, 0 }, ;
    { "FIELDNAME", "C", 10, 0 }, ;
    { "PICTURE", "C", 25, 0 }, ;
    { "SAYCOLOR", "C", 6, 0 }, ;
    { "GETCOLOR", "C", 12, 0 }, ;
    { "VALID", "C", 35, 0 }, ;
    { "WHEN", "C", 35, 0 }, ;
    { "FORMULA", "C", 50, 0 } ;
})

use getinfo new
y := len(aTestdata)
for x := 1 to y
    dbAppend()
    aeval(aTestdata[x], { | i, j | fieldput(j, i) } )
next
dbclosearea()
endif
scroll()
use getinfo new
use customer new
ddgets()
return nil

#endif // end test stub

//----- these manifest constants delineate the structure of the
//----- array to be held in each get:cargo
#define PROMPT_ROW 1
#define PROMPT_COL 2
#define PROMPT_TEXT 3
#define PROMPT_COLOR 4
#define PROMPT_FORMULA 5

```

```

function ddgets
local oGet
local nGetrow
local nGetcol
local getlist := {}
local aHold := {}
local x
local y

getinfo->(dbgtop())
dispbegin()
do while ! getinfo->(eof())

    //----- store current field contents and current field position
    aadd(aHold, { fieldget(fieldpos(getinfo->fieldname)), ;
                  fieldpos(getinfo->fieldname) } )

    //----- create the GET object
    oGet := getnew(getinfo->getrow, getinfo->getcol, makeblock( aHold ), ;
                  trim(getinfo->fieldname), trim(getinfo->picture), ;
                  if(empty(getinfo->getcolor), setcolor(), ;
                      trim(getinfo->getcolor)) )

    //----- establish VALID clause (if applicable)
    if ! empty(getinfo->valid)
        //----- determine if it is stored in the form of a code block
        //----- by checking first opening brace and pipe character
        if left(strtran(getinfo->valid, ' ', ''), 2) == "{"
            oGet:postBlock := &( trim(getinfo->valid) )
        else
            oGet:postBlock := &("{ || " + trim(getinfo->valid) + "}")
        endif
    endif

    //----- establish WHEN clause (if applicable)
    if ! empty(getinfo->when)
        //----- determine if it is stored in the form of a code block
        //----- by checking first opening brace and pipe character
        if left(strtran(getinfo->when, ' ', ''), 2) == "{"
            oGet:preBlock := &( trim(getinfo->when) )
        else
            oGet:preBlock := &("{ || " + trim(getinfo->when) + "}")
        endif
    endif
endwhile
endfunction

```

```

endif
endif

//----- store all information associated with the prompt in the
//----- cargo slot for this GET object
oGet:cargo := { getinfo->row,      ;
                getinfo->col,      ;
                trim(getinfo->prompt), ;
                getinfo->saycolor,  ;
                if(! empty(getinfo->formula), ;
                  &(trim(getinfo->formula)), NIL) ;
                }

//----- add to master GETLIST array
aadd(getlist, oGet)
getinfo->(dbskip())
enddo

//----- display all prompts, functions, and GETs
y := len(getlist)
for x := 1 to y

    //----- position cursor
    setpos(getlist[x]:cargo[PROMPT_ROW], getlist[x]:cargo[PROMPT_COL])

    //----- if there's a formula, evaluate it now
    if valtype(getlist[x]:cargo[PROMPT_FORMULA]) == "B"
        dispout( eval(getlist[x]:cargo[PROMPT_FORMULA]), ;
                if(empty(getlist[x]:cargo[PROMPT_COLOR]), NIL, ;
                  getlist[x]:cargo[PROMPT_COLOR]) )
    else
        //----- no formula, display the prompt text
        dispout(getlist[x]:cargo[PROMPT_TEXT], ;
                if(empty(getlist[x]:cargo[PROMPT_COLOR]), NIL, ;
                  getlist[x]:cargo[PROMPT_COLOR]) )
    endif

    //----- dynamically reposition the GET if getrow/getcol were not
    //----- previously specified in GETINFO.DBF
    if getlist[x]:row == 0 .and. getlist[x]:col == 0
        getlist[x]:row := row()
        getlist[x]:col := col() + 1
    endif
endfor

```

```

endif

//----- now display the GET
getlist[x]:display()
next

dispend()

readmodal(getlist)

if lastkey() != K_ESC .and. rlock()
    //----- loop through aHold array to copy values back to database
    for x := 1 to len(aHold)
        if aHold[x][1] <> NIL
            fieldput(aHold[x][2], aHold[x][1])
        endif
    next
    unlock
endif
return nil

/*
Function: MakeBlock()
Purpose: Use "detached locals" to hard-code reference to each
        element in aHold array (prevents array access error)
*/
static function makeblock(a)
local x := len(a)
return { | _1 | if(_1 == NIL, a[x][1], a[x][1] := _1) }

/*
Function: NotEmpty()
Purpose: Used for validation
Note:    Cannot be scoped as STATIC because it will be referred to
        within a code block that is compiled at run-time

Caveat:  This is included for demonstration only. I encourage you
        to write generic code block syntax operating on the GET
        object. For an example, look in GETINFO.DBF for the VALID
        clause that is used on the ZIP field.
*/
function notempty
return ! empty( getactive():varGet() )

```


الحالة الأخرى المناسبة لاستخدام وظيفة GETNEW() هي الاستعراض حيث تسمح للمستخدم بتعديل الخلايا مباشرة. يبين المثال التالي استعراض عام لقاعدة البيانات. بضغط مفتاح **Enter** تتحرك إلى حقول التعديل بواسطة وظيفة EditCell().

```
// filename: GETS22.PRG
#include "inkey.ch"

function gets22(dbf_file)
local x, browse := TBrowseDB(3, 0, 15, 79), column, key
setcursor(0)
scroll()
use (dbf_file) new
for x := 1 to fcount()
    column := TBColumnNew(field(x), fieldwblock(field(x), select()))
    browse:AddColumn( column )
next
do while key != K_ESC
    do while ! browse:stabilize() .and. ( key := inkey() ) = 0
    enddo
    if browse:stable
        key := inkey(0)
    endif
    do case
        case key == K_UP
            browse:up()
        case key == K_DOWN
            browse:down()
        case key == K_LEFT
            browse:left()
        case key == K_RIGHT
            browse:right()
        case key == K_ENTER
            editcell(browse)
    endcase
enddo
use
return nil

static function editcell(b)
//---- determine current column object out of the browse object
local column := b:getColumn(b:colPos), mget, oldcurs := setcursor(2)
//---- create a corresponding GET
mget := GetNew(Row(), Col(), column:block, column:heading, , b:colorSpec)
```

```
ReadModal( {mget} ) // must pass as an array!
setcursor(oldcurs) // reset cursor status
b:refreshCurrent()
return nil
```

مرّ معنا وظيفة () GETNEW تستلزم إنشاء كتلة شيفرة موافقة لمتغير هدف GET. وإن وظيفة () EditCell تفعل ذلك بالعمل أولاً باتجاه الخلف من هدف الاستعراض لتحديد العمود الحالي. وهذا سهل بواسطة فحص المتغير الفوري "الموضع في العمود" TBrowse:colPos ، ثم تمرير تلك النتيجة إلى وظيفة جدول العرض getColumn. وهذا يؤدي إلى وجود هدف عمود الاستعراض الذي له كتلة شيفرة مرتبطة به وتخبره مايجب عرضه. وبما أن كتلة الشيفرة هذه قد أنشئت بواسطة "كتلة الحقل" FLEDBLOCK () في كليبر 5.0 ، فإن لها التركيب المطلوب لهدف GET ذاته.

حالا يكون لدينا هدف GET ، يمرر (كعنصر في مصفوفة حرفية) إلى وظيفة READMODAL ()

```
// excerpted from STD.CH
#command READ => ReadModal(GetList) ; GetList := { }
```

يحدد أمر "قراءة" READ مسبقاً لتمرير GETLIST كمتغير. وبما أن GETLIST لا يستخدم في هذه الحالة فلا فائدة من وجوده ، لذلك لا يتعين علينا تمرير GETLIST دائماً إلى وظيفة () READMODAL. ويمكننا تمرير مصفوفة تحتوي هدفاً أو أكثر من أهداف GET ، وهذا يعمل وجوب تمرير MGET كعنصر في مصفوفة.

الوظائف المرتبطة ب: GET

هناك العديد من الوظائف المرتبطة بفئة الهدف GET ، والتي تعمل على أهداف GET وتعديلها. وقد بحثنا بعضاً منها ، وفيما يلي البقية:

الوظائف التالية ، تغيير حالة الهدف GET.

الوظيفة	الغرض منها
assign()	تعيين المحتويات المعدلة للذاكرة المؤقتة لتغير get
colorDisp()	تغيير لون Get وإعادة عرض Get على الشاشة
display()	عرض Get على الشاشة
killFocus()	إزالة المؤشر المضيء عن هدف Get
reset()	تحرير معلومات الحالة الداخلية لهدف Get
setFocus()	وضع المؤشر المضيء على هدف Get
undo()	إعادة تجهيز متغير Get لكي يعود للأصل السابق Get:original
updateBuffer()	تحديث الذاكرة المؤقتة للتحرير
varGet()	ارجاع القيمة الحالية لمتغير Get
varPut()	تجهيز متغير Get بالقيمة الممررة

تحرك الوظائف التالية المؤشر ضمن الذاكرة المؤقتة **get:buffer**:

الوظيفة	الغرض منها
end()	نقل المؤشر إلى أقصى اليمين في GET
home()	نقل المؤشر إلى أقصى اليسار في GET
left()	نقل المؤشر إلى اليسار حرف واحد
right()	نقل المؤشر إلى اليمين حرف واحد
toDecPos()	نقل المؤشر إلى اليمين المواجه للموقع العشري (decPos)
wordLeft()	نقل المؤشر إلى اليسار كلمة واحدة
wordRight()	نقل المؤشر إلى اليمين كلمة واحدة

تستخدم الوظائف التالية لتعديل وتحرير الذاكرة المؤقتة `get:buffer`:

الوظيفة	الغرض منها
<code>backspace()</code>	نقل المؤشر إلى اليسار وحذف حرف واحد
<code>delete()</code>	حذف حرف عند المؤشر
<code>delEnd()</code>	حذف من عند المؤشر إلى نهاية الذاكرة المؤقتة
<code>delLeft()</code>	حذف حرف من يسار المؤشر
<code>delRight()</code>	حذف حرف من يمين المؤشر
<code>delwordLeft()</code>	حذف كلمة من يسار المؤشر
<code>delwordRight()</code>	حذف كلمة من يمين المؤشر
<code>insert()</code>	إدخال حروف داخل الذاكرة المؤقتة
<code>overStrike()</code>	الكتابة الفوقية على الذاكرة المؤقتة

إن وظائف معظم هذه الوظائف واضحة من اسمها. وبالنسبة للمتغيرات الفورية ، كلما أردت استدعاء واحدة من هذه الوظائف يجب عليك أن تسبقها بإشارة إلى هدف `GET`. فمثلاً: تظلل الشيفرة التالية هدف `GET (G)` وتنقل المؤشر إلى أقصى موضع في الجانب الأيمن ضمنه:

```
g:setfocus( )
g:end( )
```

عرض الألوان `ColorDisp()`

تغير هذه الوظيفة ، التي أضيفت للإصدار 5.01 من كليب ، ألوان هدف `GET` ويعرضها مباشرة. فهو يشبه من الناحية الوظيفية تعيين المتغير الفوري "طيف الألوان" `get:colorSpec` وإصدار وظيفة "العرض" `(get:display)`. إن وظيفة "عرض الألوان" `(get:colorDisp)` تعني أنك تستطيع تحديد ألوان خاصة لأهداف `GET`

الخاصة بك وتطبقها على الهدف مباشرة (بدلاً من الانتظار حتى يتم تشغيل هدف GET).

يقوم المعالج الأولي بترجمة عبارة @..GET..COLOR الاختيارية إلى استدعاء لوظيفة "عرض الألوان" () get:colorDisp كما سنبين فيما يلي:

الملف الاصلي (PRG):

```
@ 21, 0 get y color "n/bg, +w/bg"
```

مخرجات المعالج الأولي (PPO):

```
SetPos( 21, 0 ) ; ;  
AAdd( GetList, _GET_( y, "y", , , ) ) ; ;  
ATail(GetList) :colorDisp("n/bg, +w/bg")
```


تحسين أداء أمر @..GET

بحسبنا سابقاً أن أمر @..GET يعالج أولاً إلى استدعاءات للوظائف التي تنشئ هدف GET وتضيفه إلى مصفوفة GETLIST. وهذه الدرجة من التركيب الهندسي المفتوح تعني أننا يمكننا الاستفادة أكثر من المعالج الأولي لإضافة تحسينات على أمر @..GET.

البنية الهيكلية لملف GETSYS.PRГ

في الإصدار 5.01 من كليبر ، تم تعديل عدد من الوظائف الرئيسة المستخدمة في معالج GET القياسي بحيث أصبحت عامة (أي يمكن استدعاؤها من وظائف في ملفات (.PRG) أخرى) وأصبح من الممكن استخدامها من قبل عمليات القراءة الخاصة بـ: GET والتي عدلتها بنفسك (والتي يمكنك تنفيذها بواسطة المتغير الفوري "قارئ" get:reader) هذه الوظائف هي:

القارئ (<oGet>) GetReader

تقرأ وظيفة () GetReader القراءة القياسية الخاصة بأهداف GET. وكافترض ، تستخدم وظيفة () ReadModol وظيفة () GetReader لقراءة أهداف GET ، ثم تستدعي هذه الأخيرة وظائف أخرى من ملف GETSYS.PRГ لقراءة هدف GET. وستفصل هذه الوظائف لاحقاً.

لاحظ أنه يمكنك إلغاء وظيفة () GetReader بواسطة تعيين كتلة شيفرة للمتغير الفوري "قارئ" get:reader. ويمكن تشكيل كتلة الشيفرة هذه لاستدعاء الوظيفة البديل الذي تختاره لوظيفة () GetReader. وسيأتي معنا مثال لذلك في بحث عبارة "رسالة" MESSAGE.

استخدام وظيفة المفاتيح (<nKey> , <oGet>) GetApplyKey

تطبق وظيفة () GetApplyKey قيمة وظيفة () inKey على هدف GET. وتغيير مفاتيح تحريك المؤشر موضع المؤشر ضمن هدف GET ، وتدخل مفاتيح البيانات في الهدف. يجب تظليل هدف GET (يركز) قبل استخدام المفاتيح. وتعالج وظيفة () GetApplyKey أيضاً ضغطات المفاتيح المرتبط بها إجراءات "مفاتيح الاستخدام السريع" Hot-Key.

وإذا أردت تنفيذ معالجة لفتاح خاص (أي: كلمة سر ، أو إدخال بيانات بأسلوب الآلات الحاسبة) فيمكنك كتابة الشكل الذي تريده من هذا الوظيفة.

ملاحظة

إذا نفذت "مسح الأهداف" CLEAR GETS بواسطة "ضبط المفاتيح" SETKEY فسيضبط المتغير الفوري "حالة الخروج" Get:exitState إلى GE_ESCAPE. وهذا يلغي هدف GET الحالي ، في النظام القياسي ، دون تعيين القيمة المعدلة ، وينهي وظيفة () ReadModal.

التدقيق السابق (<oGet>) GetPreValidate

تدقق هذه الوظيفة هدف GET لمعرفة التعديل الذي أجري عليها ، بما في ذلك القيام بتقييم "كتلة سابقة" get:perBlock (عبارة WHEN) إن وجدت. وتعيد وظيفة () get:perValidate قيمة منطقية: "حقيقي" (T.) إن كان التدقيق المسبق لهدف GET ناجحاً ، وإلا "غير حقيقي" (F.). كما يضبط المتغير الفوري "حالة الخروج" Get:exitState ليعكس نتيجة التدقيق المسبق:

الضبط	التوضيح
GE_NOEXIT	يشير إلى أن التدقيق السابق تم بنجاح ، ok للإتمام
GE_WHEN	يشير إلى أن التدقيق السابق فشل
GE_ESCAPE	يشير إلى أنه تم إصدار أمر CLEAR GETS

في نظام GET الافتراضي ، يلغي المتغير exitState المضبوط على GE_ESCAPE هدف GET الحالي وينهي وظيفة () ReadModal.

التدقيق اللاحق (<oGet>) GetPostValidate

تعين هذه الوظيفة القيمة لمتغير GET ، ثم تدققه (إن لزم الأمر) بواسطة تقييم "كتلة سابقة" get:postBlock (عبارة VALID). فهي تعيد قيمة منطقية تشير إن كان التدقيق اللاحق على هدف GET ناجحاً أم لا.

إن أصدر "مسح الأهداف" CLEAR GETS أثناء "التدقيق اللاحق" ، سيضبط المتغير الفوري "حالة الخروج" get:exitState إلى GE_ESCAPE وتعيد وظيفة () GetPostValidate قيمة "حقيقي" (.T.).

ضبط المفاتيح (<oGet>) GetDoSetKey

تنفذ هذه الوظيفة كتلة شيفرة خاصة بضبط المفاتيح SET KEY ، مع الإبقاء على سياق هدف GET الذي سبق ترميزه كما هو. وإن اسم الإجراء ورقم السطر الممرين إلى كتلة SET KEY يعتمدان على الاستدعاء الأخير لوظيفة () ReadModal (تذكر: أنه قد يكون لديك استدعاءات متداخلة للوظيفة () ReadModal إذا كان لديك أوامر قراءة متداخلة).

لقد أضيفت الوظائف الثلاثة التالية إلى ملف GETSYS.PRG في إصدار كليبر 5.2. والغرض من هذه الوظائف الثلاث هو مساعدة المطور الذي يقوم بإنشاء فئات قراءة Read متعددة (مثلاً: أمر Read متداخلة) من خلال السماح بالوصول السهل إلى مدى الملف الساكن في ملف GETSYS.PRG (وقد كانت هذه المتغيرات سابقاً لا يمكن الوصول إليها ، وفي المقابل كانت تسبب الدعر للعديد من المطورين).

قراءة ملف الشاشة READFORMAT([<bFormat>])

تمكّنك هذه الوظيفة (فقط في إصدار كليبر 5.2) من الوصول إلى ملف الشاشة الحالي (FMT) وتعديله من برنامجك دون تعديل برنامج GETSYS.PRG. لتغيير الشاشة الحالية تمرر كتلة شيفرة (<bFormat>) إلى هذه الوظيفة.

ملفات الشاشة هي من بقايا قاعدة البيانات dBASE ، ومعظم مطوري برامج كليبر 5.0 لا يستخدمونها. ولهذا ، فإن هذه الوظيفة لا يبدو أنها ستكون بالغة الفائدة لك.

إنهاء القراءة READKILL([<IKill >])

تمكّنك هذه الوظيفة من إنهاء كافة مستويات "القراءة" READ العاملة حالياً (فقط في إصدار كليبر 5.2). فتمرر قيمة منطقية (T.) لإنهاء القراءة. فهي تعيد دائماً قيمة منطقية تفيد إن كان أمر "القراءة" قد اختير لإنهائه أم لا. يفحص نظام GET هذه المجموعة بعد استدعاء أي من وظائف مفاتيح الاستخدام السريع أو WHEN أو VALID. لذلك إذا أردت استخدام وظيفة "إنهاء القراءة" READKILL() يجب أن تضعها في إحدى هذه الوظائف.

تحديث القراءة READUPDATED([<IUpdated>])

تمكّنك هذه الوظيفة (فقط كليب 5.2) من ضبط وظيفة التحديث (UPDATED). من المفيد استخدام هذه الوظيفة مع "القراءة المتدخلة"، وعندما نعين قيمة لأهداف GET يدوياً بواسطة وظيفة (get:varPut). مرور قيمة منطقية لتغيير مؤشر وظيفة التحديث (UPDATED). تعيد وظيفة READUPDATED دائماً قيمة منطقية تشير إلى ضبط وظيفة التحديث (UPDATED) الحالية.

كتابة وظيفة (GetReader) بديلة

ذكرنا آنفاً كيفية ربط رسائل بكل هدف من أهداف GET مع عبارة WHEN. ستخزن الرسالة، بدلاً من ذلك، في المتغير الفوري "شحنة" cargo الخاصة بهدف GET باستخدام المعالج الأولي.

```
#xcommand @ <row> , <col> GET <var> [PICTURE <pic>] ;  
[VALID <valid>] [ WHEN <when> ] ;  
[MESSAGE <message> ] => ;  
SetPos( <row> , <col> ) ; ;  
AAdd( GetList, _GET_( <var> , <{var}> , <pic> , ;  
      <{valid}> , <{when}> , ) :display( ) ) ; ;  
Atail(getlist):reader := { | g | MyReader(g) } ;  
[ ; Atail(getlist):cargo := <message> ]
```

لقد أضفنا عبارتين لموجه أمر GET..@ القياسي. تعين العبارة الأولى المتغير الفوري "قارئ" reader لاستدعاء وظيفة (GetReader) البديلة. وهذا ضروري لأن وظيفة (GetReader) تكون حيث نريد إدراج شيفرة عرض الرسائل.

تشير وظيفة (ATAIL) (المضافة في إصدار 5.01) إلى آخر عنصر في المصفوفة. وبما أن هدف GET هذا أضيف إلى مصفوفة getlist، فستشير وظيفة

() ATAIL إلى الهدف الجديد. وهذا يجعل الشيفرة عامة لأنه لن يكون هناك فرق مهما كان عدد أهداف GET في مصفوفة getlist.

أما العبارة الثانية ، التي تعالج أولاً في حال خصصنا عبارة "رسالة" MESSAGE فقط ، فتعين الرسالة إلى المتغير الفوري "شحنة" cargo.

توضح القائمة التالية كيفية عمل عبارة MESSAGE..@. لاحظ أن المتغير الفوري "قارئ" reder يؤسس كتلة شيفرة تستدعي وظيفة () MyReader البديلة عن وظيفة () GetReader القياسية.

```
//filename: GETS23.PRG
#ifdef CLIPPER52

#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [COLOR <color>]
    [MESSAGE <message>]

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>,
        <{when}> ):display() )
ATail(GetList):reader := { | g | MyReader(g) }
[ ; Atail(GetList):colorDisp( <color> ) ]
[ ; Atail(GetList):cargo := <message> ]

#else

#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [COLOR <color>]
    [MESSAGE <message>]

=> SetPos( <row>, <col> )
; AAdd(
```

```
        GetList,
        _GET_( <var>, <(var)>, <pic>, <{valid}>, <{when}> )
    )
    ATail(GetList):reader := { | g | MyReader(g) }
    [ ; ATail(GetList):colorDisp( <color> ) ]
    [ ; ATail(GetList):cargo := <message> ]

#endif

#define TEST          // for compiling test code -- remove if desired

#ifdef TEST          // begin test code

function gets23
local x := 0, y := 0, z := 0
local getlist := {}
scroll()
@ 10, 10 gget x message "This is the message for the first GET"
@ 11, 10 gget y          // no message
@ 12, 10 gget z message "This is the message for the last GET"
read
return nil

#endif          // end test code

#include "getexit.ch"  // necessary for the exitState constants

/*
    Function: MyReader()
    Purpose: Alternate to GetReader()
    Author: Greg Lief
    Copyright (c) 1991 Greg Lief
    Dialect: Clipper 5.01
*/
static function MyReader( get )
local mess_row := set(_SET_MESSAGE)    // row for displaying messages
// read the GET if the WHEN condition is satisfied
if ( GetPreValidate(get) )

/*
    the following IF..ENDIF, and the LOCAL declaration above,
    are the only modification to the stock GetReader() function
*/

    // show message if there is one for this GET
    // note that current SET MESSAGE row will be used
```

```
// if you prefer, you could default these to some other row
if get:cargo != NIL
  @ mess_row, 0 say padc( get:cargo, maxcol() + 1)
endif

// activate the GET for reading
get:SetFocus()

do while ( get:exitState == GE_NOEXIT )

  // check for initial typeout (no editable positions)
  if ( get:typeOut )
    get:exitState := GE_ENTER
  endif

  // apply keystrokes until exit
  do while ( get:exitState == GE_NOEXIT )
    GetApplyKey( get, Inkey(0) )
  enddo

  // disallow exit if the VALID condition is not satisfied
  if ( !GetPostValidate(get) )
    get:exitState := GE_NOEXIT
  endif

enddo

// erase message if there is one for this GET
if get:cargo != NIL
  scroll(mess_row, 0, mess_row, maxcol(), 0)
endif

// de-activate the GET
get:KillFocus()

endif

return nil
```

التوسع في استخدامات وظائف المفاتيح

() GetApplyKey البديلة

سنوسع وظيفة () GetReader بواسطة وظائف تغيير طريقة معالجة المفاتيح ضمن أهداف GET. فهل لاحظت أن الوظيفة السابقة تستدعي الوظيفة () GetApplyKey لمعالجة المفاتيح؟ ، وهذا بالطبع يفتح أمامنا طريقة جديدة كلياً لمجموعة من الامكانيات ، وبما أننا نحن الذين قمنا بكتابة شيفرة المصدر للوظيفة () GetApplyKey ، فلنا كامل الحرية في تعديلها ، وكذلك نستطيع استدعاء أي وظيفة خاصة بالمفاتيح نريدها.

إن القدرة على إمتلاك شيفرة المصدر للوظيفة () GetApplyKey وإمكانية تعديلها ، فتح لنا باباً ، لم يكن متوفراً في الإصدار السابق من كليبر. ومع أن هذه القدرة عظيمة إلا أن سوء استخدامها قد يؤدي إلى : (أ) تجاهل كافة الأحرف الصوتية أو (ب) إضافة رقم عشوائي لكل حرف تحول إلى شيفرة غير مفهومة أثناء التشغيل، وهو ما يطلق عليه encryption (وهذا غير مرغوب به).

تتيح الأمثلة الثلاثة التالية الخاصة بوظائف المفاتيح: (أ) إدخال مجموعة الحروف الملائمة (صغيرة/كبيرة) ، (ب) إدخال الخطوات ، (ج) إدخال كلمات سر مخفية. (كما سنلق النظر على GET الرياضية ، والتي تسمح للمستخدم بإدخال صيغ رياضية مثل (*45 75) ، ولكن هذا المنطق سيكون من مكونات الوظيفة () GetReader ولذلك، فلن نحتاج إلى وظيفة معالجة مفاتيح منفصلة) سنحتاج في البداية إلى أكثر من امر واحد لانقاص عامل الارباك. نظرياً من الممكن عمل طريقة لالتفاف أمر GET..@ ، وذلك للقيام بفحص كل العبارات الاختيارية ، ولكن هذا لا يستحق اضاءة الوقت.

```
// filename: KEYTEST.CH
```

```
#ifdef CLIPPER52
```

```
//----- PROPER clause
```

```
#xcommand @ <row>, <col> GGET <var>
```

```
[PICTURE <pic>]
```

```
[VALID <valid>]
```



```

[WHEN <when>]
[PROPER]
[MESSAGE <message>]
[COLOR <color>]

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>,
          <{when}>):display()
    )
ATail(GetList):reader := { | g | MyReader(g,
{ | get, key | gkeyproper(get, key) } ) }
[ ; Atail(GetList):colorDisp( <color> )]
[ ; Atail(GetList):cargo := { <message> } ]

//---- STEP clause
#xcommand @ <row>, <col> GGET <var>
[PICTURE <pic>]
[VALID <valid>]
[WHEN <when>]
[STEP [INCREMENT <step>]]
[MESSAGE <message>]
[COLOR <color>]

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>,
          <{when}>):display()
    )
ATail(GetList):reader := { | g | MyReader(g,
{ | get, key | gkeystep(get, key [, <step>]) } ) }
[ ; Atail(GetList):colorDisp( <color> )]
[ ; Atail(GetList):cargo := { <message> } ]

//---- MATH clause
#xcommand @ <row>, <col> GGET <var>
[PICTURE <pic>]
[VALID <valid>]
[WHEN <when>]
[MATH]
[MESSAGE <message>]
[COLOR <color>]

=> SetPos( <row>, <col> )
; AAdd(
    GetList,

```

```

        _GET_( <var>, <(var)>, <pic>, <{valid}>,
              <{when}>):display()
    ATail(GetList):reader := { | g | MyReader(g,
    { | get, key | getapplykey(get, key) } ) }
    ATail(Getlist):picture := "@Q"
    [ ; Atail(GetList):colorDisp( <color> )]
    [ ; Atail(GetList):cargo := { <message> } ]

```

```

//—— PASSWORD clause, using default character ("*")
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [PASSWORD]
    [MESSAGE <message>]
    [COLOR <color>]

```

```

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>,
          <{when}>):display()
    ATail(GetList):reader := { | g | MyReader(g,
    { | get, key | gkeypass(get, key) } ) }
    ATail(Getlist):picture := "@P"
    [ ; Atail(GetList):colorDisp( <color> )]
    [ ; Atail(GetList):cargo := { <message> } ]

```

```

. //—— MESSAGE clause only
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [COLOR <color>]
    [MESSAGE <message>]

```

```

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>,
          <{when}>):display()
    ATail(GetList):reader := { | g | MyReader(g,
    { | get, key | getapplykey(get, key) } ) }
    [ ; Atail(GetList):colorDisp( <color> )]
    [ ; Atail(GetList):cargo := { <message> } ]

```

```
#else
```

```
//----- PROPER clause
```

```
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [PROPER]
    [MESSAGE <message>]
    [COLOR <color>]
```

```
=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>, <{when}> )
)
ATail(GetList):reader := { | g | MyReader(g,
{ | get, key | gkeyproper(get, key) } ) }
[ ; ATail(GetList):colorDisp( <color> ) ]
[ ; ATail(GetList):cargo := { <message> } ]
```

```
//----- STEP clause
```

```
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [STEP [INCREMENT <step>]]
    [MESSAGE <message>]
    [COLOR <color>]
```

```
=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>, <{when}> )
)
ATail(GetList):reader := { | g | MyReader(g,
{ | get, key | gkeystep(get, key [, <step>]) } ) }
[ ; ATail(GetList):colorDisp( <color> ) ]
[ ; ATail(GetList):cargo := { <message> } ]
```

```
//----- MATH clause
```

```
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
```

```

[MATH]
[MESSAGE <message>]
[COLOR <color>]

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>, <{when}> )
)
ATail(GetList):reader := { | g | MyReader(g,
{ | get, key | getapplykey(get, key) } ) }
ATail(Getlist):picture := "@Q"
[ ; Atail(GetList):colorDisp( <color> ) ]
[ ; Atail(GetList):cargo := { <message> } ]

//----- PASSWORD clause, using default character ("*")
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [PASSWORD]
    [MESSAGE <message>]
    [COLOR <color>]

=> SetPos( <row>, <col> )
; AAdd(
    GetList,
    _GET_( <var>, <(var)>, <pic>, <{valid}>, <{when}> )
)
ATail(GetList):reader := { | g | MyReader(g,
{ | get, key | gkeypass(get, key) } ) }
ATail(Getlist):picture := "@P"
[ ; Atail(GetList):colorDisp( <color> ) ]
[ ; Atail(GetList):cargo := { <message> } ]

//----- MESSAGE clause only
#xcommand @ <row>, <col> GGET <var>
    [PICTURE <pic>]
    [VALID <valid>]
    [WHEN <when>]
    [COLOR <color>]
    [MESSAGE <message>]

=> SetPos( <row>, <col> )
; AAdd(

```

```

        GetList,
        _GET_( <var>, <(var)>, <pic>, <{valid}>, <{when}> )
    )
    ATail(GetList):reader := { | g | MyReader(g,
    { | get, key | getapplykey(get, key) } ) }
    [ ; ATail(GetList):colorDisp( <color> ) ]
    [ ; ATail(GetList):cargo := { <message> } ]

#endif

```

لاحظ أننا غيرنا كتل شيفرة `get:reader` قليلاً لكي نكيّف وظائف المفاتيح البديلة. أنشئت كتلة الشيفرة لتقبل متغيراً هو هدف `GET` العامل حالياً (يمرر من وظيفة `(ReadModal)`).

ومع ذلك يمرر لوظيفة `(MyReader)` متغيرين: هدف `GET` الحالي ، وكتلة شيفرة تقبل بدورها متغيرين وترسلهما إلى وظيفة المفاتيح.

استخدام وظيفة `(MyReader)` المعدلة

كان عمل هذه الوظيفة سابقاً مقتصرًا على فحص المتغير الفوري "شحنة" `cargo` للبحث عن رسائل فيه (وعرضها إن وجدت). يجب تعديل وظيفة `(MyReader)` لتقبل كتلة شيفرة المفاتيح المذكورة أعلاه. كما أن هناك تعديلات كثيرة مطلوبة لمساندة القاعدة المنطقية لعبارة "كلمة السر" `PASSWORD`. يمكن إخفاء كلمة السر بواسطة حفظها في المتغير الفوري "شحنة" `get:cargo` حتى آخر لحظة. ولكن ذلك قد يؤدي إلى مشاكل عندما نريد الدخول ثانياً إلى هدف `GET` له كلمة سر. لذلك سنضيف عند التمرير لأول مرة مصفوفة مكونة من عنصرين إلى مصفوفة "شحنة" `get:cargo` (تعرف أيضاً باسم `get:cargo[2]` الذي يحوي عنصرين هما:

١- سلسلة حرفية فارغة لوضع كلمة السر فيها.

٢- العرض الفعلي لهدف GET والذي سيستخدم لاحقاً ملء كتلة البيانات.

يجب أن يكون المتغير الفوري "شحنة" get:cargo مصفوفة ، وإن لم يكن مصفوفة فيجب تحويله إلى مصفوفة وترك حيز للرسالة بواسطة جعل العنصر الأول "صفرًا" .NIL

عندما تعود إلى هدف GET ستحتوي مصفوفة المتغير الفوري "شحنة" get:cargo عنصرين. بعد ذلك نعيد تعيين محتويات هذا المتغير ليحتوي رمز القناع الخاص بنا (هو الفراضاً نجمة) ، وإذا لم نقم بذلك ستعرض كلمة السر الفعلية على الشاشة عندما ننفذ وظيفة "ضبط التركيز" (SetFocus) ، مما سيعطي نتائج غير تلك المرجوة.

بعد استدعاء وظيفة "ضبط التركيز" (SetFocus) يجب إجراء تعديل آخر في هدف GET. إن كنا بصدد تنقيح هدفاً له كلمة سر قد تكون هناك معلومات موجودة في الذاكرة المؤقتة. وفي هذه الحالة يجب أن نقل المؤشر إلى نهاية الذاكرة لتجنب وجود فوضى غير مرئية.

يظهر التغيير الأخير بعد استدعاء وظيفة "إنهاء التركيز" (KillFocus). مازالت كلمة السر ، حتى هذه اللحظة في المتغير الفوري "شحنة" get:cargo ، ويجب علينا تعيينها إلى متغير GET باستخدام وظيفة (get:varPut). ويجب أن نعينها بعد وظيفة (KillFocus) لأن هذا الأخير يعيد عرض هدف GET ، وهذا قد يؤدي إلى نتائج غير تلك المرجوة.

كتابة وظائف المفاتيح العادية

ننسخ أولاً وظيفة (GetApplyKey) من برنامج GETSYS.PRГ في ملف PRГ. مستقل. وبما أن جميع وحدات المفاتيح البديلة الخاصة بنا سنستخدمها في الهيكل المنطقي

ذاته لتحريك المفاتيح "الأساسية" ، فسننشئ وظيفة تسمى (GKeyBasic) تحتوي تلك القاعدة المنطقية فقط.

إن كل وظيفة من وظائف المفاتيح البديلة الخاصة بنا سيتحقق إذا كان المفتاح من مفاتيح الاستخدام السريع ، فإن كان كذلك نفذ الإجراء عليه. وبعد ذلك سيعالج كل وظيفة مفاتيح عاملة معينة (بما فيها الأحرف والأرقام) وأخيراً ، إن لم يعالج المفتاح فسيمرر إلى وظيفة (GKeyBasic) كما ذكرنا أعلاه.

تعيد وظيفة (GKeyBasic) قيمة "حقيقي" (T.) إذا ميّز المفتاح ونفذ الإجراء عليه ، وإلا فسيعيد قيمة "غير حقيقي" (F.). غالباً ما تكون هذه القيمة المعادة غير هامة ، ولكن عملية إدخال السلسلة الحرفية الملائمة تحتاج لمعرفة إن كنا قد خرجنا من هدف GET باستخدام مفتاح التنقل بحيث يمكنها إعادة ضبط ذاتها.

إدخال كلمة السر (GKeyPass)

تمكّنك هذه الوظيفة من إدخال كلمة السر مخفية تماماً. عندما يدخل المستخدم كلمة السر الخاصة به تظهر على الشاشة نجوم. وحتى لو وجد أكثر من هدف GET على الشاشة ، فلن تظهر كلمة السر ولو انتقل المستخدم من هذا الهدف.

تعطل وظيفة (GKeyPass) معظم مفاتيح تحريك المؤشر (السهم الأيسر ، والسهم الأيمن ، والبداية Home والنهاية End على سبيل المثال) ، وذلك لأنها لا تعمل ضمن سياق هذا الوظيفة. والمفتاح الوحيد المسموح باستخدامه هو "مسافة للخلف" Back Spase ، طبعاً للسماح للمستخدم بتصحيح الخطأ إن وجد.

تضع هذه الوظيفة الرمز الحقيقي لكلمة السر في مصفوفة get:cargo بدلاً من وضعه في الذاكرة المؤقتة get:buffer التي تظهر فيها نجمة ("*") بدلاً منه. وهذا يوضح طبيعة نظام GET في كليبر 5.0.

مفاتيح الحروف الملائمة GKeyProper()

تتيح هذه الوظيفة إدخال صيغة الحروف الملائمة ، فبدخل الحرف الأول من كل كلمة آلياً بحرف كبير والأحرف التالية بحروف صغيرة (lower-case) ، في حين أنه لا توجد وظيفة صورة picture في dBASE أو كبير ، تسمح لك بإنجاز هذه الوظيفة. وتحمل الوظيفة GKeyProper() مؤشراً منطقياً يدعى "حرف كبير" FORCEUPPER. وعندما نحرك المؤشر في الذاكرة المؤقتة get:buffer يحدث FOREUPPER إن كان المؤشر على يمين مسافة فارغة مباشرة.

لذا وجد المتغير الفوري "موضع المؤشر" get:pos مع وظيفة GKeyProper() مفيد جداً لأنه يبين موضع المؤشر في الذاكرة المؤقتة get:buffer بالضبط.

يستخدم المؤشر المنطقي "حرف كبير" FOREUPPER أيضاً في حقل كلمة السر حيث تحول الأحرف الصغيرة إلى كبيرة. لاحظ أنك عندما تخرج من هدف GET بمفتاح التقال ، فسيعاد ضبط مؤشر FOREUPPER إلى "حقيقي" (.T.) في المرة التالية.

الوظيفة GKeyStep()

تمكّنك هذه الوظيفة من زيادة / إنقاص التواريخ والأرقام باستخدام مفاتيح الجمع والطرح. ويمكنك تعيين مقدار "زيادة" اختياري (افتراضياً سيكون (١)).

يركز عمل هذه الوظيفة في "منطقة استثناء المفاتيح" (تغير عملها). فيفحص أولاً نوع متغير GET (ويمكن الوصول إليه عن طريق المتغير الفوري get:type) لأنه يعمل فقط بالتواريخ والأرقام. ثم يغير المتغير الفوري get:changed إلى "حقيقي" (.T.) ، وهذا

هام لأننا إذا قمنا بتعديل هدف GET مباشرة بوظيفة (varGet) ووظيفة (varPut) فلا يمكن تعديل المتغير الفوري get:changed.

أما المرحلة الهامة فهي فحص عبارة "صحيح" VALID (الموجودة في المتغير الفوري get:postBlock) للتأكد من عدم تجاوزنا للحدود. أخيراً يجب علينا تحديث الذاكرة المؤقتة get:buffer يدوياً بواسطة وظيفة get:updateBuffer لأننا لم نستخدم وظيفة "إدراج" (get:Insert) أو وظيفة "إلغاء" (get:Overstrike).

```
// filename: GETS24.PRG
#include "keytest.ch"

function gets24
local w := space(20), x := space(20), y := 10, z := date(), v := 0
local year := year(z)
local getlist := {}
set(_SET_MESSAGE, 15)
setcolor('+gr/b')
scroll()
@ 10, 10 gget w password message "Please enter your password"
@ 11, 10 gget v math message "Please enter a formula"
@ 12, 10 gget x proper message "This will be proper-case"
@ 13, 10 gget y step valid y < 30 ;
    message "Use the '+' and '-' keys, and keep it under 30"
@ 14, 10 gget z step increment 7 valid year(z) == year ;
    message "Use the '+' and '-' keys, and keep it in " + str(year, 4)
read
if ! empty(w)
    @ 16, 10 say "Your password was " + trim(w) + ""
endif
inkey(0)
return nil
```

```
// filename: MYREADER.PRG
```

```
#include "getexit.ch" // necessary for the exitState constants
#include "error.ch" // necessary for custom error handler
#include "inkey.ch"

#define MESSAGE 1
#define PASSWORD cargo[2, 1]
#define PASSWIDTH cargo[2, 2]
#define PASSCHAR ""
```



```
#xtranslate UsingPassword() => ;
    oGet:picture != NIL .and. substr(oGet:picture, 1, 2) == "@P"

#xtranslate UsingMath() => ;
    oGet:picture != NIL .and. substr(oGet:picture, 1, 2) == "@Q"

function MyReader( oGet, bKeyproc )
local mess_row := set(_SET_MESSAGE)    // row for displaying messages
//---- the next five items are needed for MATH gets
local newhandler
local oldhandler
local oldblock
local cExpr
local nOldvalue
// read the GET if the WHEN condition is satisfied
if ( GetPreValidate(oGet) )
    // show message if there is one for this GET
    // note that current SET MESSAGE row will be used
    // if you prefer, you could default these to some other row
    if oGet:cargo != NIL .and. ! empty(oGet:cargo[MESSAGE])
        @ mess_row, 0 say padc( oGet:cargo[MESSAGE], maxcol() + 1)
    endif

    if UsingPassword()

        //---- if we have already been in this GET,
        //---- the cargo array will already contain 2 items
        if oGet:cargo != NIL .and. len(oGet:cargo) == 2
            oGet:varPut(padl(replicate(PASSCHAR, len(oGet:PASSWORD)), ;
                oGet:PASSWIDTH))
        else
            //---- determine if cargo is an array or not
            if oGet:cargo == NIL
                oGet:cargo := { NIL }    // account for message slot
            endif

            //---- add subarray to cargo holding the following items:
            //---- 1) empty string to hold the contents of the password
            //---- 2) width of GET for padding later
            aadd(oGet:cargo, { "", len(oGet:varGet()) } )

        endif
    endif

    //---- activate the GET for reading
    oGet:SetFocus()
```



```
//---- force cursor to end of the GET for password-style
//---- if there is already data in this GET
if UsingPassword() .and. ! empty(oGet:buffer)
  oGet:end()
endif

do while ( oGet:exitState == GE_NOEXIT )
  // check for initial timeout (no editable positions)
  if ( oGet:typeOut )
    oGet:exitState := GE_ENTER
  endif

  // if this is a MATH get, we must change the block to point
  // at a character string, and change the picture clause
  // to scroll accordingly
  if UsingMath()
    nOldvalue := eval(oGet:block)
    cExpr := padr(ltrim(str(nOldvalue)), 100)
    oldblock := oGet:block
    oGet:block := { | _1 | if(_1 == NIL, cExpr, cExpr := _1) }
    oGet:picture := "@K@S" + ltrim(str(len(oGet:buffer)))

    // the get:type instance variable must be changed
    // from "N" to "C", and the only way I could do it
    // was to remove and reset focus
    dispbegin()
    oGet:killFocus()
    oGet:setFocus()
    dispend()
  endif

  // apply keystrokes until exit
  do while ( oGet:exitState == GE_NOEXIT )
    eval(bKeyproc, oGet, inkey(0))
  enddo

  // if this was a MATH get, time for presto-chango back to numeric
  if oldblock != NIL
    oGet:block := oldblock
    oGet:picture := "@Q"
    newhandler := { | e | blockhead(e, oldhandler) }
    oldhandler := errorblock(newhandler)
    if ! empty(oGet:buffer)
      begin sequence
        oGet:varPut( eval("&('{ || " + trim(oGet:buffer) + "}") ) )
        oGet:changed := .t.
      recover
    endif
  endif
enddo
```

```
        oGet:varPut( nOldvalue )
    end sequence
    // we must change get:type from "C" back to "N" for
    // validation, and as noted above, the only way I
    // found to do it was to kill and reset focus
    dispbegin()
    oGet:killFocus()
    oGet:setFocus()
    dispend()
endif
errorblock(oldhandler)
endif

// disallow exit if the VALID condition is not satisfied
if ( !GetPostValidate(oGet) )
    oGet:exitState := GE_NOEXIT
endif
enddo
// remove message for this GET if there was one
if oGet:cargo != NIL .and. ! empty(oGet:cargo[MESSAGE])
    scroll(mess_row, 0, mess_row, maxcol(), 0)
endif

// de-activate the GET
oGet:KillFocus()
//----- if password style entry was used, time to actually assign the value
if UsingPassword()
    oGet:varPut(padr(oGet:PASSWORD, oGet:PASSWIDTH))
    oGet:changed := .t.
endif
endif
return nil

static function blockhead(oError, bOldhandler)
if oError:genCode == EG_NOVAR .or. oError:genCode == EG_SYNTAX
    alert("Syntax error in formula")
    break
endif
return eval(bOldhandler, oError)

// filename: GKBASIC.PRG

#include "inkey.ch"
#include "getexit.ch"

function GKeyBasic(oGet, nKey)
```

```

local ret_val := .t.
do case
  case ( nKey == K_UP )
    oGet:exitState := GE_UP
  case ( nKey == K_SH_TAB )
    oGet:exitState := GE_UP
  case ( nKey == K_DOWN )
    oGet:exitState := GE_DOWN
  case ( nKey == K_TAB )
    oGet:exitState := GE_DOWN
  case ( nKey == K_ENTER )
    oGet:exitState := GE_ENTER
  case ( nKey == K_ESC )
    if ( Set(_SET_ESCAPE) )
      oGet:undo()
      oGet:exitState := GE_ESCAPE
    endif
  case ( nKey == K_PGUP )
    oGet:exitState := GE_WRITE
  case ( nKey == K_PGDN )
    oGet:exitState := GE_WRITE
  case ( nKey == K_CTRL_HOME )
    oGet:exitState := GE_TOP
  // both ^W and ^End terminate the READ (the default)
  case (nKey == K_CTRL_W)
    oGet:exitState := GE_WRITE
  otherwise
    ret_val := .f.
endcase
return (ret_val)
// filename: GKPASS.PRG

#include "inkey.ch"
#include "getexit.ch"
#define PASSWORD    cargo[2, 1]
#define PASSWIDTH   cargo[2, 2]
#define PASSCHAR    ""

function gkeypass(oGet, nKey)
local cKey
local bKeyBlock := SetKey(nKey)
//----- check for hot key
if bKeyBlock != NIL
  GetDoSetKey(bKeyBlock, oGet)
else
  do case

```

```

case (nKey == K_BS)
  oGet:BackSpace()
  oGet:PASSWORD := substr(oGet:PASSWORD, 1, len(oGet:PASSWORD)-1)
case (nKey >= 32 .and. nKey <= 255)
  cKey := chr(nKey)
  //----- keep the real string in oGet:cargo and display asterisks
  oGet:PASSWORD += cKey
  oGet:Overstrike(PASSCHAR)
  if (oGet:typeOut .and. !Set(_SET_CONFIRM) )
    if ( Set(_SET_BELL) )
      ?? Chr(7)
    endif
    oGet:exitState := GE_ENTER
  endif
otherwise
  gkeybasic(oGet, nKey)
endcase
endif
return nil

```

// filename: GKPROPER.PRG

```

#include "inkey.ch"
#include "getexit.ch"

```

```

function gkeyproper(oGet, nKey)
static forceupper := .t.
local cKey
local bKeyBlock := SetKey(nKey)
//----- check for hot key
if bKeyBlock != NIL
  GetDoSetKey(bKeyBlock, oGet)
else
  do case
    case (nKey == K_INS)
      Set( _SET_INSERT, ! Set(_SET_INSERT) )
      setcursor( if(set(_SET_INSERT), 3, 1) )
    case (nKey == K_CTRL_U)
      oGet:Undo()
      forceupper := .t. // reset flag for next time
    case (nKey == K_HOME)
      oGet:Home()
      forceupper := .t.
    case (nKey == K_END)
      oGet:End()
      forceupper := .f.
  endcase
endif

```

```

case (nKey == K_RIGHT)
    oGet:Right()
    forceupper := (substr(oGet:buffer, oGet:pos - 1, 1) == " ")
case (nKey == K_LEFT)
    oGet:Left()
    //----- if we just backspaced to the start of a word or the string,
    //----- we must reset the flag to force next character upper-case
    forceupper := (oGet:pos==1 .or. ;
        substr(oGet:buffer, oGet:pos-1, 1) == " ")
case (nKey == K_CTRL_RIGHT)
    oGet:WordRight()
    forceupper := .t. // cause we are at the beginning of a word
case (nKey == K_CTRL_LEFT)
    oGet:WordLeft()
    forceupper := .t. // cause we are at the beginning of a word
case (nKey == K_BS)
    oGet:BackSpace()
    /* if we just backspaced to the start of a word, or
       the start of the string, we must reset the flag to
       force the next character to be upper-case
    */
    forceupper := (oGet:pos==1 .or. ;
        substr(oGet:buffer, oGet:pos-1, 1) == " ")
case (nKey == K_DEL)
    oGet>Delete()
case (nKey == K_CTRL_T)
    oGet:DelWordRight()
case (nKey == K_CTRL_Y)
    oGet:DelEnd()
case (nKey == K_CTRL_BS)
    oGet:DelWordLeft()
    forceupper := .t. // cause we are at the beginning of a word
case (nKey >= 32 .and. nKey <= 255)
    cKey := Chr(nKey)
    if nKey == 32 /* space bar */
        forceupper := .t. // next char will be uppercase
    elseif nKey > 64
        if forceupper
            if nKey > 96 .and. nKey < 123
                ckey := chr(nKey - 32)
            endif
            forceupper := .f.
        endif
    endif
    if ( Set(_SET_INSERT) )
        oGet:Insert(cKey)

```



```
    else
        oGet:Overstrike(cKey)
    endif
    if (oGet:typeOut .and. !Set(_SET_CONFIRM) )
        if ( Set(_SET_BELL) )
            ?? Chr(7)
        endif
        oGet:exitState := GE_ENTER
    endif
otherwise
    //----- reset FORCEUPPER flag if navigation key was pressed
    if GKeyBasic(oGet, nKey)
        forceupper := .t.
    endif
endcase
endif
return nil
```

// filename: GKSTEP.PRG

```
#include "inkey.ch"
#include "getexit.ch"
```

```
function gkeystep(oGet, nKey, nStep)
local cKey
local oldvalue
local bKeyBlock := SetKey(nKey)
//----- check for hot key
if bKeyBlock != NIL
    GetDoSetKey(bKeyBlock, oGet)
else
    do case
        case (nKey == K_INS)
            Set(_SET_INSERT, ! Set(_SET_INSERT) )
            setcursor( if(set(_SET_INSERT), 3, 1) )
        case (nKey == K_CTRL_U)
            oGet:Undo()
        case (nKey == K_HOME)
            oGet:Home()
        case (nKey == K_END)
            oGet:End()
        case (nKey == K_RIGHT)
            oGet:Right()
        case (nKey == K_LEFT)
            oGet:Left()
        case (nKey == K_CTRL_RIGHT)
```

```

oGet:WordRight()
case (nKey == K_CTRL_LEFT)
oGet:WordLeft()
case (nKey == K_BS)
oGet:BackSpace()
case (nKey == K_DEL)
oGet:Delete()
case (nKey == K_CTRL_T)
oGet:DelWordRight()
case (nKey == K_CTRL_Y)
oGet:DelEnd()
case (nKey == K_CTRL_BS)
oGet:DelWordLeft()
case (nKey >= 32 .and. nKey <= 255)
cKey := chr(nKey)
//----- test for step entry on numerics and dates
if cKey $ '-+' .and. oGet:type $ "ND"
if nStep == NIL
nStep := 1
endif
oGet:changed := .t.
oldvalue := oGet:varGet()
if cKey == "-"
oGet:varPut(oldvalue - nStep)
else
oGet:varPut(oldvalue + nStep)
endif
if oGet:postBlock != NIL .and. ! eval(oGet:postBlock, oGet)
oGet:varPut(oldvalue)
endif
oGet:updateBuffer()
else
if ( Set(_SET_INSERT) )
oGet:Insert(cKey)
else
oGet:Overstrike(cKey)
endif
if (oGet:typeOut .and. ISet(_SET_CONFIRM) )
if ( Set(_SET_BELL) )
?? Chr(7)
endif
oGet:exitState := GE_ENTER
endif
endif
otherwise
gkeybasic(oGet, nKey)

```

```
endcase  
endif  
return nil
```

قراءة موقوتة

تستخدم وظيفة () MyReader وظيفة INKEY(0) لمتابعة ضغط المفاتيح ، ثم تمرر الضغطة إلى وظيفة المفاتيح الملائمة. وسنستخدم الحلقة التالية مكان وظيفة INKEY(0).

```
do while ( nkey := inkey( ) ) == 0  
enddo
```

لقد حصلنا على النتائج ذاتها مع اختلاف واحد وهو أن الحلقة مفتوحة تماماً ، فعلى سبيل المثال يمكننا تعيين عبارة أخرى لقطع هذه الحلقة ولم يضغط أي مفتاح:

```
nstart := seconds( )  
ntimeout := 60  
do while ( nkey := inkey( ) ) == 0 .and. seconds( ) - nstart < ntimeout  
enddo
```

بعد ذلك يمكننا إدراج قاعدة منطقية إضافية بعد الحلقة لمعرفة إذا تم الخروج بواسطة مفتاح أو "توقيت الخروج":

```
// if we broke the loop with a keystroke , process it  
if nkey := 0  
    eval(bkeyproc, oGet, nkey)  
// we timed out.... process timeout event if one was specified  
else  
    // perform timeout action  
endif
```

سنفترض أننا نريد في معظم الأحيان إدخال مفتاح [Esc] في الذاكرة المؤقتة ليسبب الخروج من "قراءة" READ ، لكننا نريد إجراء عملية أخرى في بعض الأحيان (مثل: إقفال البيانات CLOSE DATA). فنستطيع إعادة صياغة القاعدة المنطقية IF..ENDIF لمعرفة إذا تم تعيين حالة خروج ، أو أجريت العملية الافتراضية (إدخال مفتاح [Esc]).

```
// if we broke the loop with a keystroke , process it
if nkey := 0
    eval(bkeyproc, oGet, nkey)
// otherwise process timeout event if one was specified
elseif exit_event := NIL
    eval(exit_event)
    // otherwise perform default action
else
    keyboard chr(K_ESC)
endif
```

وظيفة توقيت الخروج GFTIMEOUT()

يجب أن نعين لوظيفة MyReader() قيمتي مدة الإلتظار وزمن الخروج. ومع أنه يمكن استخدام المتغير الفوري "قاريء" get:reader لتمرير هاتين القيمتين كمتغيرين إضافيين إلا أن العملية ستصبح معقدة جداً. لذلك ، سنضيف وظيفة "توقيت الخروج" GFTIMEOUT() في آخر ملف PRG الذي يحتوي وظيفة MyReader().

تحتوي وظيفة GFTIMEOUT() مصفوفة ساكنة من عنصرين. يشتمل العنصر الأول على عدد ثواني الإلتظار لضغطة مفتاح ، بينما يشتمل الثاني على كتلة شيفرة اختيارية تحدد زمن الخروج في حال انتهاء وقت الإلتظار ، إن تشكيل هذه الوظيفة يشبه تشكيل وظيفة SET(). لاحظ أنها تعيد أيضاً ضبط المصفوفة الساكنة إذا لم نمررها أية متغيرات.

```
function gftimeout(ntime, val)
static settings_ := { 60000, }
local ret_val
if ntime == NIL // reset array
    settings_ := { 60000, }
else
    ret_val := settings_[ntime]
    if val := NIL
        settings_[ntime] := val
    endif
endif
return ret_val
```

موجّه المعالج الأولي Preprocessor Directive

أما الخطوة الأخيرة فهي توفير موجّه المعالج الأولي ليعين هاتين القيمتين بشكل صحيح:

```
// manifest constants for use with GFTimeout( )
#define SECONDS 1
#define EXIT_EVENT 2
#xcommand READ TIMEOUT <seconds>
    [EXITEVENT <exitevent>] =>
    gftimeout( SECONDS, <seconds> )
    [ ; gftimeout(EXIT_EVENT, <{exit_event}>) ]
    ; ReadModal( gellist )
    ; gellist := { }
    ; gftimeout( )
```

تستدعى وظيفة GFTimeOut() أولاً لضبط عدد ثواني الانتظار ، ثم تستدعى ثانية ، إذا كنت تعين EXITEVEN الاختياري ، لضبط زمن الخروج أيضاً. بعد ذلك تستدعى وظيفة READModal() لبدء التشغيل. وبعد الانتهاء من القراءة READ تستدعى وظيفة GFTimeOut() مرة أخرى لمسح هاتين القيمتين.

لقد صمم هذا التركيب ليكون مرناً ، فيمكننا التنقل بين موجّه READ TIMEOUT وأمر "قراءة" READ حسبما نريد. فإذا كنا نتوقع أن تكون كافة القراءات مؤقتة ، فيمكننا تحسين الأداء باستدعاء وظيفة GFTimeOut() مباشرة في البرنامج وباستخدام أمر READ.

```
gftimeout(SECONDS, 60) // time out after one minute of inactivity
gftimeout(EXIT_EVENT, { | | cleanup( ) } )
//
//
read
```


التدقيق الكلي

لحسن الحظ ، فإن المصفوفة GETLIST تجعل التدقيق الكلي سهلاً. فبدلاً من تثبيت شيء ما داخل الوظيفة () GetReader البديلة الخاصة بنا ، سنستدعي وظيفة () GFVlid ، بعد الرجوع من وظيفة () ReadModal. ستلف هذه الوظيفة مصفوفة GETLIST وتفحص كل عنصر فيه بحثاً عن عبارة VALID. فإذا وجدت عبارة VALID في هدف GET (أو في أي متغير فوري post Block) فستفحص هذه العبارة.

مثال

يوضح المثال التالي "القراءة الموقوتة" timedread و "التدقيق الكلي" total validation . سيتم تحديد زمن القراءة بعد (١٠) ثوان من عدم إرسال الرموز، وتستدعي وظيفة الخروج () Au_Revoir. يجب أن يكون آخر هدفين من GET أكبر من الصفر ، لذلك يجب محاولة الخروج من القراءة READ بضغط مفتاحي [W]- [Ctrl] فوقهما.

ومن أجل راحتك ، فقد قمنا بإضافة ملف RMAKE في أسطوانة شيفرة المصدر المرفق بهذا الكتاب ، والذي يقوم بتجميع وربط هذا المثال. كل ما هنالك أن تقوم بطباعة الأمر RMAKE NEWREADS.RMK.

```
// filename: GETS25.PRG
#include "inkey.ch"
#include "keytest.ch"
#include "newreads.ch"
```

```
function gets25
local w := space(20)
local x := space(20)
local y := 10
local z := date()
```

```
local zz := { 0, 0}
local nyear := year(z)
local getlist := {}
set(_SET_MESSAGE, 18)
setcolor('+gr/b')
scroll()
@ 1, 12 say "I will terminate after 10 seconds of keyboard inactivity"
@ 10, 10 gget w password message "Please enter your password"
@ 11, 10 gget x proper message "This will be proper-case"
@ 12, 10 gget y step valid y < 30 ;
    message "Use the '+' and '-' keys, and keep it under 30"
@ 13, 10 gget z step increment 7 valid year(z) == nyear ;
    message "Use the '+' and '-' keys, and keep it in " + str(nyear, 4)
@ 14, 10 gget zz[1] message "Must be greater than zero!" ;
    valid { | g | notzero(g) }
@ 14, 30 say "Note: this must be greater than zero!"
@ 15, 10 gget zz[2] message "Must be greater than zero!" ;
    valid { | g | notzero(g) }
@ 15, 30 say "Note: this must be greater than zero!"
read timeout 10 exitevent au_revoir() validation
if ! empty(w)
    @ 17, 10 say "Your password was " + w
endif
return nil
```

```
static function au_revoir
keyboard chr(K_ESC)
return nil
```

```
static function notzero(get)
local ret_val := .t.
if get:varGet() <= 0
    get:colorDisp("W/R,+W/R")
    ret_val := .f.
endif
return ret_val
```

```
/*
Function: MyReader()
Purpose: Alternate to GetReader()
Author: Greg Lief
Copyright (c) 1991-3 Greg Lief
Dialect: Clipper 5.01
*/
```

```
#include "getexit.ch" // necessary for the exitState constants
```

```

#define MESSAGE      1
#define PASSWORD      cargo[2, 1]
#define PASSWIDTH      cargo[2, 2]
#define PASSCHAR      "*"

#xtranslate UsingPassword() => ;
    oGet:picture != NIL .and. substr(oGet:picture, 1, 2) == "@P"

/*
    Function: MyReader()
    Purpose: Alternate to GetReader()
    Copyright (c) 1991-3 Greg Lief
*/
static function MyReader( oGet, bKeyproc )
local mess_row, oldcursor, nkey, nstart, ntimeout, bAction

// read the GET if the WHEN condition is satisfied
if ( GetPreValidate(oGet) )
    mess_row := set(_SET_MESSAGE)

// show message if there is one for this GET
if oGet:cargo != NIL .and. ! empty(oGet:cargo[MESSAGE])
    @ mess_row, 0 say padc( oGet:cargo[MESSAGE], maxcol() + 1)
endif
if UsingPassword()
    //----- if we have already been in this GET,
    //----- the cargo array will already contain 2 items
    if oGet:cargo != NIL .and. len(oGet:cargo) == 2
        oGet:varPut(padr(replicate(PASSCHAR, len(oGet:PASSWORD)), ;
            oGet:PASSWIDTH))
    else
        //----- determine if cargo is an array or not
        if oGet:cargo == NIL
            oGet:cargo := { NIL }      // account for message slot
        endif

        //----- add subarray to cargo holding the following items:
        // 1) empty string to hold the contents of the password
        // 2) width of GET for padding later
        aadd(oGet:cargo, { "", len(oGet:varGet()) } )
    endif
endif

//----- activate the GET for reading

```

```
oGet:SetFocus()

//----- force cursor to end of the GET for password-style
//----- if there is already data in this GET
if UsingPassword() .and. ! empty(oGet:buffer)
  oGet:end()
endif

do while ( oGet:exitState == GE_NOEXIT )
  // check for initial typeout (no editable positions)
  if ( oGet:typeOut )
    oGet:exitState := GE_ENTER
  endif

  // apply keystrokes until exit
  do while ( oGet:exitState == GE_NOEXIT )
    nstart := seconds()
    ntimeout := gftimeout(SECONDS)
    do while ( nkey := inkey() ) == 0 .and. seconds() - nstart < ntimeout
      enddo

    //----- if we broke the loop with a keystroke, process it
    if nkey != 0
      eval(bKeyproc, oGet, nkey)
    //----- otherwise process timeout event if one was specified
    elseif ( bAction := gftimeout(EXIT_EVENT) ) != NIL
      eval(bAction)
    //----- otherwise stuff an ESC into the buffer
    else
      keyboard chr(K_ESC)
    endif
  enddo

  // disallow exit if the VALID condition is not satisfied
  if ( !GetPostValidate(oGet) )
    oGet:exitState := GE_NOEXIT
  endif

enddo

// remove message for this GET if there was one
if oGet:cargo != NIL .and. ! empty(oGet:cargo[MESSAGE])
  scroll(mess_row, 0, mess_row, maxcol(), 0)
endif
```

```

// de-activate the GET
oGet:KillFocus()

//----- if password style entry was used, time to actually assign the value
if UsingPassword()
    oGet:varPut(padr(oGet:PASSWORD, oGet:PASSWIDTH))
endif

endif

return nil
//----- end of function MyReader()

function gftimeout(nitem, val)
static settings_ := { 60000, }
local ret_val
if nitem == NIL    // reset array
    settings_ := { 60000, }
else
    ret_val := settings_[nitem]
    if val != NIL
        settings_[nitem] := val
    endif
endif
return ret_val

```

وظيفة GET العامة

تستخدم وظيفة (BoxGet) العديد من خصائص كليبر 5 ، بما في ذلك هدف GET .
 إن هذه الوظيفة عامة فهي ترسم مربعاً وتعرض رسالة توجيه وتحديد المتغير. وفيما يلي القاعدة اللغوية لأمر BoxGet التي يحددها المستخدم.

```

BOXGET <var> PROMPT <prompt> [ PICTURE <pict> ] [ VALID <valid> ] ;
      [ BOXCOLOR <boxcolor> ] [ COLOR <color> ] [ ROW <row> ] ;
      [ COLUMN <column> ] [ NORESTORE ] [ RESTOREALL ] [ DOUBLE ]

```


المتغيرات المطلوبة

<Var> هو اسم المتغير المطلوب.

<prompt> هو تعبير مكون من رموز تمثل رسالة التوجيه التي ستعرض.

المتغيرات الاختيارية

<pict> هو تعبير مكون من رموز تمثل عبارة PICTURE التي ستستخدم لتغيير
.GET

<valid> هو عبارة VALID التي ستستخدم لتغيير .GET

<boxcolor> هو تعبير مكون من رموز تمثل اللون الذي سيلون به المربع. وإن لم تعينه
سيستخدم اللون الحالي.

<color> هو تعبير مكون من رموز تمثل لون هدف GET . بحيث تعين ذلك بصيغة
<enhanced> , <standard> " حيث سيستخدم <enhanced> عندما يظل
GET. وإذا لم يعين ستستخدم مجموعات الألوان القياسية والمحسنة.

<row> "الصف" و <column> "العمود" هما الصف العلوي والعمود الأيسر حيث
سيعرض المربع. وإذا لم تعين هذين المتغيرين سيعرض وظيفة () BoxGet المربع في وسط
الشاشة أفقياً أو عمودياً أو معاً.

<title> هو تعبير مكون من رموز. إذا عينته سيعرض في وسط الصف العلوي للمربع.

توجه عبارة NORESTORE وظيفة () BoxGet بعدم استعادة الشاشة الفارغة
(النموذج) تلقائياً عند الخروج. تستخدم هذه العبارة لعرض سلسلة من أهداف GET
في الشاشة ، فإذا عينت ستضاف محتويات الشاشة الفارغة إلى مجموعة لاستعادتها لاحقاً
بواسطة عبارة .RESTOREALL.

تستعيد عبارة RESTOREALL كافة أجزاء الشاشة المخزنة في مجموعة النافذة المذكورة أعلاه .

تؤدي عبارة DOUBLE إلى رسم مربع بإطار مزدوج. كقيمة افتراضية يرسم المربع بخط واحد.

إعادة اقيمة

للتعبء وظيفة () BoxGet أية قيمة لأنها تعالج <var> مباشرة بواسطة كتلة شيفرة.

ملاحظات

من أول الأشياء التي قد تندمش منها ، هي كيف يمكن لوظيفة مثل () BoxGet أن تعمل على المتغيرات التي تم إعلانها محليا local في البرنامج ذو المستوى الاعلى (وبالخصوص عندما تعتبر أن () BoxGet ترجع بدون قيمة).

انظر بعناية للتركيب اللغوية للأمر المعرف من قبل المستخدم والموجود في ملف شيفرة المصدر. لقد تم إنشاء هدف GET باستخدام الوظيفة () GETNEW. إن هذه الوظيفة تتطلب استرجاع كتلة الشيفرة من أجل المتغير GET. وفيما يلي مثال على كتلة استرجاع من أجل المتغير MNAME:

```
{ | x | IF(PCOUNT( ) = 0 , mname, mname := x ) }
```

إن كتلة شيفرة الاسترجاع هذه تقبل متغير اختياري واحد فقط. فإذا تم تمرير المتغير لكتلة الشيفرة ، فإن قيمة المتغير تعيين للمتغير GET.

إن الشيء المهم ، الذي يجب تذكره هو أن كليبر 5.0 ، لاتعدل مباشرة المتغير GET، بل تعدل من خلال كتلة شيفرة الاسترجاع المتوافقة معها.

عبارة VALID

على عكس وظيفة () _GET_ الداخلية ، لا تسمح وظيفة () GETNEW بتمرير عبارة VALID لربطها بهدف GET. والحل الأمثل هو أن يحوّل المعالج الأولي عبارة VALID إلى كتلة شيفرة ثم تمرر كتلة الشيفرة إلى وظيفة () BoxGet ، بعدها نعين كتلة الشيفرة هذه إلى المتغير الفوري postBlock بهدف GET.

لحساب الإحداثيات الصحيحة للمربع ، يجب تحديد طول متغير GET. ويمكن ذلك بفحص طول عبارة "صورة" PICTURE الخاصة به ولكن قد لا تعين عبارة PICTURE. كما لا يمكننا الاعتماد دائماً على عبارة PICTURE لأنها قد تحتوي راسمة منطقية مضللة (مثل : "@!" , "@R").

لذلك ، يجب تحديد طول الذاكرة المؤقتة بهدف GET لحساب الإحداثيات ، ولكن يجب تشغيل هدف GET بواسطة وظيفة "ضبط التركيز" setFocus قبل فحص الذاكرة المؤقتة لأن المتغير الفوري get:buffer لا يوجد عندما يكون هدف GET في حالة تشغيل. تشغل العبارات التالية هدف GET ، وتحدد طول الذاكرة المؤقتة ، ثم توقف تشغيل الهدف.

```
oGet:setFocus( )
getlength := len(oGet:buffer)
oGet:killFocus( )
```

يجب إيقاف تشغيل الهدف لأن المربع لم يرسم بعد (تذكر .. بأنه يجب أن نعرف طول الذاكرة المؤقتة قبل أن نفكر في رسم المربع). من السهل إعادة تعيين المتغيرين الفوريين "الصف" get:row و "العمود" get:col إلى الموضع المناسب ولكن يجب أولاً معرفة مكان ظهور المربع.

```
// filename: GETS26.PRG
#include "box.ch"
```

```
#xcommand DEFAULT <param> TO <value> ==> ;
IF <param> == NIL ; <param> := <value> ; END
```

```

#xcommand BOXGET <var>
    PROMPT <prom>
    [ BOXCOLOR <boxcolor> ]
    [ PICTURE <pict> ]
    [ VALID <valid> ]
    [ COLOR <color> ]
    [ ROW <row> ]
    [ COLUMN <column> ]
    [ <norest:NORESTORE> ]
    [ <restall:RESTOREALL> ]
    [ <double:DOUBLE> ]
    =>
    BoxGet(<prom>, <row>, <column>,
        getnew( maxrow() + 1, maxcol() + 1,
        { |_grumpy | if(PCOUNT() = 0, <var>, <var> := _grumpy ) }, ;
        <(var)>, <pict>, <color> ), ;
        <{valid}>, <boxcolor>, <.norest.>, <.restall.>, <.double.> )

#define TEST

// the following stub program demonstrates how to use this beast

#ifdef TEST

function gets26
local nx := 0
local cx := space(6)
local dx := ctod("")
local cphone := "5035881815"
dispbox(0, 0, maxrow(), maxcol(), replicate(chr(176), 9))
dispbox(0, 30, 3, 49, B_SINGLE + ' ', "W/RB")
@ 1, 32 say "BOXGET() Samples" color "+GR/B"
@ 2, 34 say "By Greg Lief" color "+W/RB"
boxget nx prompt "Default Location"
boxget nx prompt "Numeric w/ picture" row 4 column 0 norestore picture '9999'
boxget cx prompt "Character" boxcolor "+GR/N" norestore row 12 column 30
boxget cx prompt "Character, double box" double restoreall row 15 column 0
boxget dx prompt "Date with validation (! EMPTY)" picture "99/99/99" ;
    valid (! EMPTY(dx)) boxcolor "W/R" color "+W/R,+W/R"
boxget cphone prompt "Phone # template" picture "@R (999) 999-9999" ;
    color "+W/B,+W/B"
return nil

#endif

#define TOP    nRow

```



```

#define LEFT  nCol
#define BOTTOM nRow + 2
#define RIGHT nCol + len(cPrompt) + 4 + nGetlength

function BoxGet(cPrompt, nRow, nCol, oGet, cValid, cBoxcolor, ;
               Inorestore, IRestall, IDouble)
//----- declare static array to hold screen shots
static aBoxstack_ := {}
local nX
local nGetlength
local aSettings := { setcursor(1), ;           // old cursor size
                     row(), col(), ;           // old cursor position
                     setcolor(cBoxcolor), ;    // old color
                     set(_SET_SCOREBOARD, .f.) } // I hate SCOREBOARD!!
/*
   Determine length of GET variable. We do this by verifying the length of
   the GET buffer. This is necessary not only in the event that no PICTURE
   clause was used, but also because we cannot always trust the PICTURE
   clause (e.g., "@!" and "@R"). Note that the GET must be activated with
   the setFocus() method, because the g:buffer instance variable only exists
   when the GET is active.
*/
oGet:setFocus()
nGetlength := len(oGet:buffer)
oGet:killFocus()
//----- truncate long prompts to fit on screen
cPrompt := substr(cPrompt, 1, maxcol() - 4 - nGetlength)
default nRow to int( maxrow() / 2)
default nCol to int( maxcol() - 3 - len(cPrompt) - nGetlength ) / 2
//----- if neither NORESTORE and RESTOREALL options were specified, add
the
//----- affected portion of screen so we can restore it upon exit
if ! Inorestore .AND. ! IRestall
    aadd(aSettings, savescreen(TOP, LEFT, BOTTOM, RIGHT) )
else
    //----- otherwise, add to box stack for restoration later (RESTOREALL)
    aadd(aBoxstack_, { TOP, LEFT, BOTTOM, RIGHT, ;
                      savescreen(TOP, LEFT, BOTTOM, RIGHT) } )
endif
@ TOP, LEFT, BOTTOM, RIGHT box if(IDouble, B_DOUBLE, B_SINGLE) + ''
setpos(TOP + 1, LEFT + 2)
//----- assign the postBlock instance variable for the VALID clause
oGet:postBlock := cValid
dispout( cPrompt )

```



```

oGet:row := row()
oGet:col := col() + 1
readmodal( { oGet } )

//----- restore all screens if RESTOREALL was specified
if !Restall .and. ! empty(aBoxstack_)
  for nX = len(aBoxstack_) TO 1 STEP -1
    restscreen(aBoxstack_[nX, 1], aBoxstack_[nX, 2], aBoxstack_[nX, 3], ;
      aBoxstack_[nX, 4], aBoxstack_[nX, 5])
  next
  aBoxstack_ := {}
elseif ! !Norestore
  restscreen(TOP, LEFT, BOTTOM, RIGHT, aSettings[6])
endif

//----- restore all other environmental aspects
setcursor(aSettings[1])           // cursor size
setpos(aSettings[2], aSettings[3]) // cursor position
setcolor(aSettings[4])           // color setting
set(_SET_SCOREBOARD, aSettings[5]) // argh...
return nil

```

ملاحظة

بعد عدة شهور اكتشفت ، أن هناك طريقة بديلة لتحديد طول الذاكرة المؤقتة لهدف GET دون عرضه على الشاشة.

```

nLength := len( transform( eval(oGet:block) , ;
  if(oGet:picture == NIL, '', oGet:picture) ) )

```

لكننا استخدمنا الطريقة المنطقية القديمة ("الصف" و "العمود") لنبين أن هناك خيارات أخرى متوفرة.

الخلاصة

بحثنا في هذا الباب أهداف GET ومصفوفة GETLIST. ويمكنك الآن استخدام عبارة WHEN ، كما يمكنك تجهيز أهداف GET بواسطة وظيفة (GETNEW). وقد عرضنا أيضاً كيفية الاستفسار من المتغيرات الفورية وتغييرها بسهولة ، وكيفية استدعاء الوظائف وتغيير حالة المؤشر أو هدف GET. ويمكنك تنفيذ القراءة الموقوتة والتدقيق الإجمالي ضمن برنامجك. فلا حذّ لخصائص نظام GET في كليب 5.0.

القسم الثالث

معالجة الأخطاء والأهداف الخاصة بالأخطاء

معالجة الأخطاء والأهداف الخاصة بالأخطاء

تمهيد

قد يظهر في برنامج كليبر نوعان من الأخطاء: (أ) الأخطاء المنطقية أو اللغوية في برنامجك (مثل: عدم توافق نوع البيانات ، والأخطاء الطباعية) ، (ب) الحالات الخارجة عن إرادة المبرمج وسيطوته (بعض الأخطاء التي قد يرتكبها المستخدم كأن يفصل التيار عن الطابعة أثناء طباعة التقرير بغير قصد ، أو يلغي قواعد بيانات ضرورية).

يجب ، طبعاً ، تصحيح الأخطاء من النوع (أ) ، وسنبحث في هذا الباب كيفية تصحيح الأخطاء من النوع (ب) بحيث يصبح البرنامج الذي نصممه أقرب للكمال.

برنامج ERRORSYS.PRГ في كليبر Summer'87

يتضمن كليبر Summer'87 على برنامج ERRORSYS.PRГ الذي يمكّننا من التعامل مع عدة أخطاء شائعة أثناء التشغيل (مثل أخطاء فتح الملفات ، وأخطاء نظام التشغيل DOS ، والمشاكل المرتبطة بالشبكة).

اشتمل هذا البرنامج على ستة وظائف صغيرة تستند في تسميتها إلى فئة الخطأ:

(Expr_Error) : أخطاء في التعبير (عدم توافق نوع البيانات).

(Undef_Error) : أخطاء غير محددة.

(Misc_Error) : أخطاء متنوعة.

(Open_Error) : أخطاء في فتح الملفات (ملفات مفقودة ، عدم توفر الذاكرة الكافية للمعالجة).

(Db_Error : أخطاء في قاعدة البيانات.

(Print_Error : أخطاء في الطابعة.

عندما يظهر خطأ أثناء التشغيل تُستدعى الوظيفة المرتبط به لمعالجته ، وتكرر متغيرات متنوعة ، حسب نوع الخطأ ، إلى الوظيفة لتوفير معلومات أكثر (مثل اسم الإجراء ، ورقم السطر ...) فعلى سبيل المثال: عند محاولة فتح ملف ليس موجوداً في قاعدة البيانات ستُستدعى وظيفة (Open_Error) وتعرض الرسالة التالية في الصف العلوي من الشاشة:

Proc MAIN line 1, open error CUSTOMER.DBF (2) Retry? (Y/N)

إن برنامج ERRORSYS.PRG الموجود في كليبر 5.0 أعم من ذلك. فعندما يظهر خطأ أثناء التشغيل يُصدر برنامج كليبر هدف خطأ Error object يمرر إلى وظيفة معالجة الأخطاء العامل حالياً. ولا توجد في كليبر 5.0 الوظائف الستة المذكورة أعلاه حيث يمكن بهذه الطريقة كتابة نموذج معالجة الأخطاء لأجزاء معينة من برنامجك.

سنناقش في البحث التالي مفهوم تركيب "بداية التسلسل..نهاية التسلسل" (BEGIN SEQUENCE..END SEQUENCE).

بداية التسلسل..نهاية التسلسل

يمكن استخدام التركيب BEGIN SEQUENCE..END SEQUENCE في كليبر 5.0 لتحديد البرامج المعقدة. ويساعد هذا التركيب في توضيح تركيبات عبارات معقدة مثل التالية:

```
begin sequence
  do while x < 20
    x++
    if 1Test .and. nNum > 15
```

```
for y := 1 to 50
  do while ctest := "GAMMA"
    cAnswer := SomeFunc( x, y, "ABC" )
    if cAnswer == "TROUBLE"
      break
    else
      doWith( cAnswer )
    endif
  enddo
next y
endif
enddi
end sequence

if cAnswer == "TROUBLE"
  React( x, y, "ABC" )
endif
```

إذا أصدرت عبارة "إيقاف" BREAK في أي مكان ضمن تركيب "بداية التسلسل..نهاية التسلسل" فإنها تجبر البرنامج على الانتقال إلى العبارة التي تلي "نهاية التسلسل" مباشرة. وفي مثالنا أعلاه نفحص (cAnswer) (أو أي مؤشر آخر نحدده ضمن تركيب "بداية التسلسل..نهاية التسلسل") لمعرفة إذا كان قد حدث خطأ فعلاً أم أننا خرجنا من التركيب بشكل طبيعي.

عبارة "إصلاح" RECOVER

يمكن تحسين تركيب "بداية التسلسل" Begin Sequence بإضافة عبارة "إصلاح" RECOVER. فإذا وجدت هذه العبارة وتوقف البرنامج BREAK في نقطة ضمن تركيب "بداية التسلسل..نهاية التسلسل" ، فسينتقل مسار البرنامج إلى العبارة التي بعد عبارة "إصلاح" RECOVER مباشرة. وإذا لم يصدر "إيقاف" BREAK فسيقدم مسار البرنامج من "بداية التسلسل" إلى عبارة RECOVER ، ثم يقفز مباشرة إلى العبارة التي تلي "نهاية التسلسل" (وبالتالي تجاوز أي برنامج إصلاح).

إن تركيب "Begin Sequence..Recover..End Sequence" " بداية التسلسل..إصلاح .. نهاية التسلسل" مشابه لتركيب "IF..ELSE..ENDIF" ، والفارق الوحيد هو أننا في التركيب الثاني ، نأخذ الفروع (IF) أو الآخر (ELSE) ، بينما في تركيب "بداية التسلسل" يمكننا تنفيذ جزء من الفرع الأول ثم ننفذ الفرع الثاني بأكمله. ويوضح المثال التالي هذه القاعدة المنطقية.

```
begin sequence
  // open a database
  // operation on the file
recover
  err_msg("Could not process file")
end sequence
```

استخدام عبارة RECOVER

يمكن إعادة صياغة عبارة "إصلاح" RECOVER بإضافة عبارة "استخدام" USING الاختيارية مما يمكننا من تمرير متغيراً إليها ، ثم تنفيذ عبارة إيقاف " BREAK (مثل: <BREAK something>).

ومع أن هذا المتغير يكون عادة هدف خطأ Error object ، ولكن يمكن أن يكون أي شيء نريده. وسنرى كيفية استخدام عبارة "استخدام الإصلاح" RECOVER USING في مثال بعد بحث المتغيرات الفورية المرتبطة بهدف الخطأ Error object.

برنامج ERRORSYS.PRГ في كليبر 5.0

إن وظيفة "نظام الخطأ" () ERRORSYS هي أول ما يُستدعى في كليبر 5.0 كما في كليبر Summer'87. ولكن مهمته في كليبر 5.0 هي: إرسال كتلة شيفرة كمعالج فعال للخطأ. والشيفرة الافتراضية له هي:

```
proc ErrorSys( )  
  ErrorBlock( { | e | DefError(e) } )
```

() ErrorBlock هي وظيفة جديدة في كليبر 5.0 يركب (أو "يرسل") كتلة شيفرة كمعالج فعال للخطأ. وكما ذكرنا أعلاه ، عندما يظهر خطأ أثناء التشغيل سيصدر كليبر 5.0 هدف خطأ Error object ويمرره كمتغير إلى كتلة الشيفرة المرسلة بوظيفة () ErrorBlock.

() DefError هي وظيفة توجد في ملف برنامج ERRORSYS.PRГ المصمم لمعالجة الأخطاء بطريقة مشابهة للوظائف الستة في كليبر Summer'87.

هدف الخطأ Error Object

عندما يظهر خطأ أثناء التشغيل ينشئ كليبر هدف خطأ Error Object ، تركيبه أبسط من تركيب هدف جدول الاستعراض TBrows وهدف عمود الاستعراض TBColumn وهدف GET لأنه لا ترتبط به أية وظائف. بل يحتوي متغيرات فورية تحتوي معلومات خاصة بالخطأ الذي حدث. وهذه المتغيرات هي:

e:args : مصفوفة بقيم العامل أو الوظيفة. يستخدم في أخطاء القيم وتكون قيمته "الصفري" NIL إن لم يستخدم.

e:canDefault : يبين إن كان هناك إصلاح الفراضي أم لا. فإذا أعاد برنامج معالجة الخطأ قيمة "حقيقي" (T.) ، يمكن استخدام الإصلاح الافتراضي الذي يحدده النظام الفرعي المرتبط بالخطأ.

e:canRtry : يبين إمكانية المحاولة ثانية بعد حدوث الخطأ. فإذا أعاد برنامج معالجة الخطأ قيمة "حقيقي" (T.) ، يمكن المحاولة ثانية. ويحدد إمكانية المحاولة النظام الفرعي المرتبط بالخطأ أيضاً.

e:cenSubstitute : يبين إن كان بالإمكان استبدال نتيجة جديدة بالقيمة بعد حدوث الخطأ. ويمكن إعادة القيمة من برنامج معالجة الخطأ كقيمة بديلة.

e:cargo : حيز يحدده المستخدم لتخزين المعلومات المطلوبة (مثل هدف الاستعراض وهدف عمود الاستعراض وهدف GET). يمكن استخدام هذا المتغير لتمرير معلومات من أحد معالجي الأخطاء إلى آخر عند ربط برامج المعالجة ببعضها.

e:description : هي سلسلة حرفية تصف حالة الخطأ. وهي مفيدة جداً.

e:filename : اسم الملف الذي ظهر فيه الخطأ. ويستخدم فقط في الأخطاء المرتبطة بالملف.

e:genCode : رمز خطأ عام في برنامج كليبر حسب ما هو في ملف (ERROR.CH). فإذا احتوى هذا المتغير قيمة الصفر ، وهي حالة نادرة الحدوث ، فيدل هذا على أن الخطأ خاص بالنظام الفرعي.

وفيما يلي قائمة بكافة رموز الأخطاء (أخذت من ملف الترويسة ERROR.CH:

EG_ARG	1
EG_BOUND	2
EG_STROVERFLOW	3
EG_NUMOVERFLOW	4
EG_ZERODIV	5
EG_NUMERR	6

EG_SYNTAX	7
EG_COMPLEXITY	8
EG_MEM	11
EG_NOFUNC	12
EG_NOMETHOD	13
EG_NOVAR	14
EG_NOALIAS	15
EG_NOVARMETHOD	16
EG_BADALIAS	17
EG_DUPALIAS	18
EG_CREATE	20
EG_OPEN	21
EG_CLOSE	22
EG_READ	23
EG_WRITE	24
EG_PRINT	25
EG_UNSUPPORTED	30
EG_LIMIT	31
EG_CORRUPTION	32
EG_DATATYPE	33
EG_DATAWIDTH	34
EG_NOTABLE	35
EG_NOORDER	36
EG_SHARED	37
EG_UNLOCKED	38
EG_READONLY	39
EG_APPENDLOCK	40

يتضح معنى معظم هذه الرموز من أسمائها ، وسنشرح فيما يلي الرمزين اللذين أضيفا في إصدار كليبر 5.01a:

EG_DUPALIAS : يظهر هذا الخطأ إذا حاولنا فتح قاعدة بيانات بواسطة نسخة مكافئة في حالة استخدام.

EG_BADALIAS : يظهر هذا الخطأ عند وجود خطأ في أسماء النسخ المكافئة. يجب أن يبدأ اسم النسخة المكافئة بحرف ويحتوي رموز رقمية و / أو حروفاً تحتها خط.

e:operation : وصف لأسم العملية أو المتغير الذي سبب الخطأ (إن وجد). يطبق هذا المتغير على أنواع معينة من الأخطاء فقط ، لكنه مفيد عند توفره.

e:osCode : رقم شيفرة خطأ نظام التشغيل. فإذا لم يجد ما يوافق رمز خطأ نظام التشغيل ، فإن هذا المتغير سيحتوي على الصفر. وهذا هام جداً لتوضيح السبب في عدم فتح أو إنشاء الملف.

e:severity : رقم يشير إلى خطورة الخطأ. وفيما يلي أرقام الدرجات من ملف
: ERROR.CH

ES_WHOCARES	0
ES_WARNING	1
ES_ERROR	2
ES_CATASTROPHIC	3

يصدر كليب 5.0 أهداف خطأ من الدرجة (١) و (٢). فإذا استخدمت وظيفة () ERRORNEW لإنشاء أهداف خطأ خاصة بك فيمكنك استخدام درجتي الخطورة الأخرتين حسب الحالة. فمثلاً إذا اكتشف البرنامج أثناء التشغيل خطأ قد يسبب تعطيل وتوقف البرنامج وتخريب الشبكة فيمكنك تعيين الدرجة (٣) لهذه الحالة. التحذير الوحيد الذي يصدره كليب 5.0 أثناء التشغيل حالياً هو "ذاكرة صغيرة" "memory low".

e:subCode : رمز خطأ خاص بالنظام الفرعي. إذا احتوى هذا المتغير "الصفر" فهذا يعني أن النظام الفرعي الذي يظهر فيه الخطأ لا يعين رموز أخطاء.

e:subSystem : وصف للنظام الفرعي الذي أصدر الخطأ (DBFNTX, BASE).

e:tries : عدد يبين عدد المحاولات التي تمت على العملية.

سنبين فيما يلي محتويات المتغيرات الفورية بعد ظهور نوعين من الخطأ. أولاً ، سنحاول استخدام ملف AB.DBF غير موجود.

args	Nil
canDefault	.t.
canRetry	.t.
canSubstitute	.f.
description	Open error
filename	TEST.DBF
genCode	21 (EG_OPEN)
operation	
osCode	2
subCode	1001
subSystem	DBF
tries	1

ثانياً سنحاول زيادة السلسلة الحرفية:

args	{ "string" }
canDefault	.f.
canRetry	.f.
canSubstitute	.t.
description	Argument error
filename	<none>
genCode	1 (EG_ARG)
operation	"++"
osCode	0
severity	2
subCode	1086
subSystem	BASE
tries	0

وباختصار ، فإن هدف الخطأ ، مصفوفة من معلومات الأخطاء تمرر إلى معالج الأخطاء. والآن وبعد أن تعرفنا على هدف الخطأ ، دعنا نشاهد كيف يمكننا استخدامها عملياً.

كتابة برنامجنا الخاص لمعالجة الأخطاء

ذكرنا أعلاه أن العبارة التالية في ملف شيفرة المصدر :ERRORSYS.PRG

```
ErrorBlock( { | e | DefError(e) } )
```

ترسل كتلة الشيفرة كمعالج للخطأ. ويمكن استخدام هذه العبارة في مكان آخر في برنامجنا لإرسال معالج الأخطاء المعدل حسب رغبتنا.

باستخدام تركيب Begin Sequence..(RECOVER)..End Sequence مع وظيفة () ErrorBlock يمكننا كتابة وظائف خاصة تفحص فقط أنواعاً خاصة من الأخطاء في أجزاء من البرنامج.

ويوضح المثال التالي هذا المبدأ. سنفتح عدة قواعد بيانات ونعمل عليها. وسنفترض أن مستخدم البرنامج "سيحذف" الملفات ، وبالتالي لن نفترض وجودها عندما نحاول فتحها.

باستخدام المتغيرات الفورية لهدف الخطأ في عبارة "استخدام الإصلاح" RECOVER USING ، يمكننا فحص سبب الخطأ.

```
// filename: RECOVER.PRG

#include "error.ch"
function main
local bOldError := errorblock( { | e | FileCheck(e, bOldError) } )
local oError
begin sequence
    use customer new
    use invoices new
    use vendors new
    // code to process the files would go here
recover using oError
    if oError:osCode == 2
        Alert(oError:filename + " not found" )
    elseif oError:osCode == 4
        Alert("Insufficient file handles")
    endif
end sequence
errorblock( bOldError ) // reset previous error handler
return nil
function FileCheck(e)
if e:genCode == EG_OPEN
    break e
endif
return .f.
```


نستخدم أولاً وظيفة `ErrorBlock()` لحفظ إشارة إلى معالج الأخطاء الحالي `(bOldError)` ونرسل معالج الأخطاء الخاص بنا (الذي يستدعي وظيفة `(fileCheck())`). وتعيد الوظيفة `ErrorBlock()` حالته الخاصة ، مما يسهل علينا إعادة ضبط معالج الخطأ السابق فيما بعد.

بعد ذلك نخطط عبارات `USE` الخاصة بتركيب `Begin Sequence..Recover..End`. لاحظ أن عبارة "إصلاح" `RECOVER` تشمل عبارة "استخدام" `USING` حتى تقبل تمرير هدف خطأ وإرجاعه بواسطة وظيفة `FileCheck()`. ما يمرر الهدف هو عبارة `BREAK` الموجودة في وظيفة `FileCheck()`.

يفحص برنامج الإصلاح المتغير الفوري `e:osCode` لتحديد طبيعة الخطأ ، نبحث في هذا المثال عن ملف مفقود (٢) ومعالجة غير كافية للملف (٤). والمهم في حالة حدوث مثل هذا الخطأ هو أن مسار البرنامج سيتجاوز برنامج الإصلاح لمعالجة هذه الملفات.

نعيد ضبط معالج الخطأ السابق في أسفل وظيفة `Main()` بواسطة `ErrorBlock()`. تفحص وظيفة `FileCheck()` المتغير الفوري `e:genCode` لمعرفة إن كان الخطأ يتعلق بفتح ملف (`EG_OPEN`). فإن كان كذلك تصدر عبارة `BREAK` ، وتمرر `E` في عبارة `BREAK` هدف الخطأ كمتغير إلى عبارة `RECOVER USING`. لاحظ أن هدف الخطأ مرئي فقط ضمن كتلة شيفرة معالج الخطأ.

ربط معالجات الأخطاء ببعضها

تعالج وظيفة (DefError) في ملف برنامج ERRORSYS.PPG أخطاء معينة قابلة للتصحيح (غالباً ما تكون متعلقة بالشبكة) ، كما أنها تعرض رسالة "تنبيه" (ALERT) التي تنبؤنا بوجود الخطأ. ولتجنب إعادة كتابة هذه الوظيفة كلما أردنا إرسال معالج خطأ جديد مؤقتاً ، سنربط معالج الخطأ العادي الخاص بنا بمعالج الخطأ الافتراضي. والقاعدة هي: كلما حدث خطأ أثناء التشغيل ، يمرر هدف الخطأ إلى معالج الخطأ السابق (الافتراضي).

يوضح المثال التالي هذه الإجراءات. وسنراجع فيه إشارة إلى معالج الخطأ الحالي ثم نمررها كمتغير ثانٍ إلى معالج الخطأ العادي الخاص بنا. ثم نمرر أمر تحكم إلى معالج الخطأ السابق بواسطة تقيم كتلة الشيفرة ، بدلاً من إعادة قيمة "غير حقيقي" (F.).

```
// filename: CHAIN.PRG

#include "error.ch"

function main
local bOldError := errorblock()
local e
errorblock( { | e | printerror(e, bOldError) } )
begin sequence
    set device to print
    @ 1,1 say date()
    @ 2,1 say crash
    set device to screen
end sequence
errorblock( bOldError ) // reset previous error handler
return nil

function printerror( error, oldhandler )
local nchoice
if error:genCode == EG_PRINT
    nchoice := alert("The printer is not responding;" + ;
        "Ensure it is on-line & ready.", ;
```

```
        { "Retry", "Quit" } )
if nchoice == 1
    return .t.
else
    break
endif
endif
return eval(oldhandler, error)
```

فإذا كان الخطأ متعلقاً بالطابعة ، فإن وظيفة (PrintError) تزود المستخدم برسالة خطأ تحتوي معلومات أكثر تفصيلاً من معالج الخطأ الافتراضي في كليبر. أما إن لم يكن الخطأ متعلقاً بالطابعة فستمرره وظيفة (PrintError) إلى معالج الخطأ السابق.

أمثلة على معالج الخطأ العادي

اختيرت الأمثلة التالية من شيفرة المصدر لـ: Reporter.

مفتاح فهرس غير صحيح Invalid Index Key

عند السماح للمستخدم باختيار ملفات فهرس يجب التحقق إن كان ملف الفهرس يطابق قاعدة البيانات الحالية. وقد اخترنا مفتاح الفهرس من ترويسة ملف NTX. ويجب الآن تقييمه للتحقق من صحته. فإذا حدث خطأ أثناء عملية التقييم ، لن تعين قيمة لـ: RET_VAL. لاحظ أن معالج الخطأ العادي يبحث عن وظائف مفقودة قد تكون جزءاً من مفتاح الفهرس ، ويتيح للمستخدم متابعة الإجراء بعد تحذيره.

```
function locksmith(cfile, ckey)
local oldhandler := errorblock( )
local b
local ret_val
if ! empty(ckey)
    errorblock( { | e | BogusKey(e, oldhandler) } )
    b := &("{ | | + ckey + "}")
    use (cfile) shared
    begin sequence
```

```

        ret_val := eval(b)
    end sequence
    use
    errorblock(oldhandler)
endif
return (ret_val != NIL)

static function BogusKey(e, oldhandler)
if e:genCode == EG_NOVAR .or. e:genCode == EG_NOALLIAS
    err_msg( { 'This Index cannot match the current database' } )
    break
elseif e:genCode == EG_NOFUNC
    error_msg( { "WARNING: This index requires the " + e:operation + " function" } )
    return .t.
endif
return eval(oldhandler, e)

```

ملفات مفقودة Missing Files

لتحقق في المثال التالي من وجود ملفات في المسار المخزن في دليل البيانات . لا يبحث
معالج الخطأ العادي عن ملف مفقود أو مسار لحسب ، بل يبحث عن سواقة مفقودة .

```

local newhandler := { | e | file_error(e, oldhandler) }
local oldhandler := errorblock(newhandler)
for x := 1 to z
    begin sequence
        use ( files_[x, PATH] + files_[x, FILENAME] ) new
    recover
        // file not found in specified path ... check current directory
        // if it's there, than scrub the path from the data dictionary
        if file( files_[x, FILENAME] )
            files_[x, PATH] := ''
            use (files_[x, FILENAME] ) new
        else
            err_msg( { "Missing file " + files_[x, FILENAME] } )
            return .f.
        endif
    end sequence
    fields_ := dbstruct()
    use
    //
    // codes to verify field information in this file

```

```
//
next
errorblock(oldhandler)
return .t.

#define MISSING_FILE 2
#define MISSING_PATH 3
#define MISSING_DRIVE 15

static function File_Error(e, oldhandler)
if e:genCode == EG_OPEN .and. ;
    ( e:osCode == MISSING_PATH .or. e:osCode == MISSING_FILE .or. ;
      e:osCode == MISSING_DRIVE )
    break
endif
return eval(oldhandler, e)
```

خطأ "تجاوز الحد" في جدول الاستعراض TBrowse

من المعروف أن الاستعراض TBrowse ينهار بحدوث خطأ "تجاوز الحد Limit Exceeded" فإذا جهزنا عدداً كبيراً من الأعمدة لمعالجتها (أي تجاوزنا حد كليبر الخاص بمجموعات الرموز ([64k]) . ومن الصعب متابعة مسافة عرض العمود التراكمية. إن هذا الخطأ يمكن تمييزه فله كشف بالثوابت التي تمثله ، لذلك يمكن كتابة معالج خطأ عادي بسيط ولكن فعال لكشفه.

```
newhandler := { | e | limitcheck(e, oldhandler) }
oldhandler := errorblock(newhandler)
begin sequence
    // code to build columns and execute the Tbrowse
and sequence
errorblock(oldhandler)
return nil
```

```
static function limitcheck(e, oldhandler)
if e:genCode == EG_LIMIT
    err_msg({ "You are attempting to view too many fields at once" , ;
              "Please unselect some of your fields before vewing" } )
    break
endif
return eval(oldhandler, e)
```


التحقق من حالة إقفال السجل الحالي

لا يمكن استخدام وظيفة (RLOCK) في كليبر للتحقق من أن سجلاً مقفول أم لا ، لأنه سيحاول إقفاله. وفي حالة الرغبة في فحصه دون إقفاله ، يمكننا استخدام وظيفة (IsLocked) التي تستخدم معالجة الخطأ الخاص بكليبر 5.0. تحاول هذه الوظيفة تعيين محتويات الحقل الأول فيه (بواسطة وظيفة (FIELDPUT) وهذا يتطلب أن يكون السجل مقفلاً. فإن لم ينجح التعيين ، نستنتج أن ذاك السجل ليس مقفلاً.

```
function Islocked
local ret_val := .t.
local oldhandler := errorblock( { | e | break(NIL) } )
begin sequence
  fieldput(1, fieldget(1))
recover
  ret_val := .f.
end sequence
errorblock(oldhandler)
return ret_val
```

تقييم كتل الشيفرة المعرفة من قبل المستخدم

يبين المثال التالي تركيباً متداخلاً "Begin Sequence..End Sequence" وهذا هام لأن هناك مرحلتان قد يحدث فيهما الخطأ : (أ) عندما ننشئ كتلة الشيفرة ، و (ب) عند تقييم كتلة الشيفرة.

```
newhandler := { | e | blockhead(e, oldhandler) }
oldhandler := errorblock(newhandler)
begin sequence
  y := &( "{ | | " + flds_[x] [FORMULA] + "}" )
  // ensure no type mismatches within this block
  begin sequence
    z := eval(y)
  end sequence
end sequence

function blockhead(e, oldhandler)
```

```
if e:genCode == EG_SYNTAX .or. e:genCode == EG_ARG
    err_msg( { "Error in code block syntax" } )
    break
elseif e:genCode == EG_NOVAR
    err_msg( { "Missing variable " + e:operation, ;
              "Please specify alias in formula" } )
    break
elseif e:genCode == EG_NOALIAS
    error_msg( { "Incorrect alias (" + e:operation + ") in formula" } )
    break
endif
return eval(oldhandler, e)
```

إنشاء أهداف الخطأ باستخدام الوظيفة ERRORNEW()

تمكنا وظيفة ERRORNEW() من إنشاء أهداف الخطأ الخاصة بنا التي نمررها بعد ذلك إلى معالج الخطأ الحالي لمعالجتها بالطريقة ذاتها كخطأ "عادي" أثناء التشغيل.

لستخدم في مثال وظيفة searchPath() أدناه وظيفة ERRORNEW() لإنشاء هدف خطأ إن لم يمرر اسم الملف كمتغير. ثم يرسل هدف الخطأ إلى معالج الخطأ الحالي.

```
// filename : ERRORNEW.PRG
#include "error.ch"

function main
    scroll()
    ? "CLIPPER.EXE found in " + searchpath('clipper.exe')
    inkey(0)
    searchpath()
    return nil

function SearchPath(cfile)
    local cpath := getenv("PATH") + ";" , ret_val := []
    local ptr
    local oErr
    //----- if filename was not
    if empty(cfile)
        oErr := ErrorNew()
        oErr:severity := ES_ERROR
        oErr:genCode := EG_ARG
```

```

oErr:args := {cfile}
oErr:canRetry := .F.
oErr:canDefault := .F.
oErr:canSubstitute := .F.
oErr:description := "Filename not passed to SearchPath()"
//----- next two are optional -- just wanted to show their use
oErr:subsystem := "UDF"
oErr:subCode := 1234
Eval(ErrorBlock(), oErr)
BREAK
else
  ptr := at(";", cpath)
endif
do while ptr > 0
  ret_val := substr(cpath, 1, ptr - 1)
  ret_val := ret_val + if(right(ret_val, 1) != '\', '\', "")
  if file(ret_val + cfile)
    exit
  else
    ptr := at(";", cpath := substr(cpath, ptr + 1) )
    ret_val := []
  endif
enddo
return ret_val

```

مع أننا قد لا نحتاج هذه الوظيفة ، إلا أنه من المفيد معرفة أن هذه الإمكانية متوفرة تحت يديك. وتفيد هذه الوظيفة كثيراً لكشف الأخطاء ضمن نطاق وحدة جزئية من برنامج كليبر.

وظيفة التنبيه () ALERT

تتوفر هذه الوظيفة في الإصدار 5.01 من كليبر وهي مفيدة جداً في عرض رسائل الأخطاء على الشاشة أمام المستخدم. يستخدم برنامج ERRORSYS.PRG هذه الوظيفة ، ولكن يمكنك استخدامها في أي جزء من البرنامج.

تمتاز هذه الوظيفة بثلاث خصائص : (أ) من السهل استدعاؤه ، (ب) يتحقق إن كانت سواقة الشاشة الكاملة قد ركبت أم لا ويعمل وفقاً لذلك ، (ج) لا يؤثر أبداً على بيئة

العمل الحالية ، حتى أنه يحفظ قيمة آخر مفتاح في الذاكرة المؤقتة للوحة المفاتيح (المعادة بواسطة وظيفة (LASTKEY).

القاعدة اللغوية لوظيفة "التنبيه" (ALERT) هي:

Alert(<cMessage> [, <aOptions>] [, <cColor>])

حيث <cMessage> هي الرسالة التي ستظهر على الشاشة ("File not found" ، "Printer off-line" الخ) وهما على التوالي الملف غير موجود والطابعة مفصولة. يمكن أن تشمل الرسالة على أكثر من سطر واحد ، ونفصل بين الأسطر بواسطة (؛).

<aOptions> هي مصفوفة من الاختيارات ليختار المستخدم واحداً منها (مثل: "Quit" ، "Retry" ..). فإذا لم نمرر هذا المتغير ، سيستخدم الوظيفة خياراً وحيداً ("موافق") (OK) ، ولكن من الأفضل تمرير مصفوفة الرسائل.

<cColor> هي مجموعة رموز تمثل لون المربع.

وإذا كانت سَواة الشاشة الكاملة موجودة وفي حالة تشغيل ، يرسم وظيفة ALERT() مربعاً ويعرض فيه الاختيارات بشكل "أزرار". وإن استدعاء الوظيفة:

nChoice := Alert("Hard Disk Metledl" , { "Abort", "Continue" , "Retry" })

يولد المربع التالي:

Hard Disk Metledl		
Abort	Continue	Retry

تنعكس ألوان الأحرف والخلفية عند الأضرار. أما إن لم تكن سؤاقة الشاشة الكاملة موجودة فتكون رسالة التنبيه () ALERT كالتالي:

Hard Disk Melted! (Abort, Continue Retry)

ويتنظر البرنامج أن يضغط المستخدم أحد أحرف الاختيارات ("A" أو "C" أو "R"). سيحتوي nChoice في مثالنا رقم العنصر الذي اختاره المستخدم. فإذا اخترنا "Continue" أي المتابعة ، فسيحتوي nChoice الرقم (٢).

وظيفة معالج الخطأ () ErrorHandler (جديد في إصدار كليبر 5.2)

من المحتمل وجود خطأ في برنامج معالج الخطأ. ويوضح المثال التالي المشكلة:

```
errorblock( { | e | TrapError(e) } )
? x // causes run-time error because x is undefined
return nil
```

```
function TrapError(e)
? x // Causes another run-time error!
return nil
```

يسبب ذلك استدعاء وظيفة () TrapError "كشف الخطأ" مرات ومرات. ثم يكتشف المعالج الأولي الداخلي في كليبر أنه يجب إيقاف الوظيفة فيصدر رسالة "خطأ لا يمكن إصلاحه" (Unrecoverable error 650).

تُصدر الوظيفة الجديدة في كليبر 5.2 () ERRORINHANDLER خطأ "إخفاق في إصلاح الخطأ" الذي يعطينا اسم وحدة البيانات ورقم السطر فلا نضيع الوقت في تتبع مكان الخطأ.

سنعيد كتابة المثال أعلاه باستخدام وظيفة () ERRORINHANDLER ومتغير ساكن لمعرفة إن كنا دخلنا معالج الخطأ أم لا.

```
function main
errorblock( { | e | TrapError(e) } )
? x // causes run-time error because X is undefined
return nil
```

```
function TrapError(e)
static lAlready := .f.
if lAlready
    ErrorInHandler( )
endif
lAlready := .t.
// Your error-handling code would go right here
lAlready := .f.
return .t.
```

تسجيل الأخطاء على الأسطوانة

تتوفر في أسطوانة برنامج المصدر (source code diskette) نسخة معدلة من برنامج ERRORSYS.PRG. وتشمل التغييرات إمكانية معالجة أخطاء الطابعة (بما فيها توجيه العمل أثناء التشغيل إلى ملف الأسطوانة) ، وتسجيل كافة المعلومات المتعلقة بالخطأ في ملف بصيغة نص. تشمل هذه المعلومات:

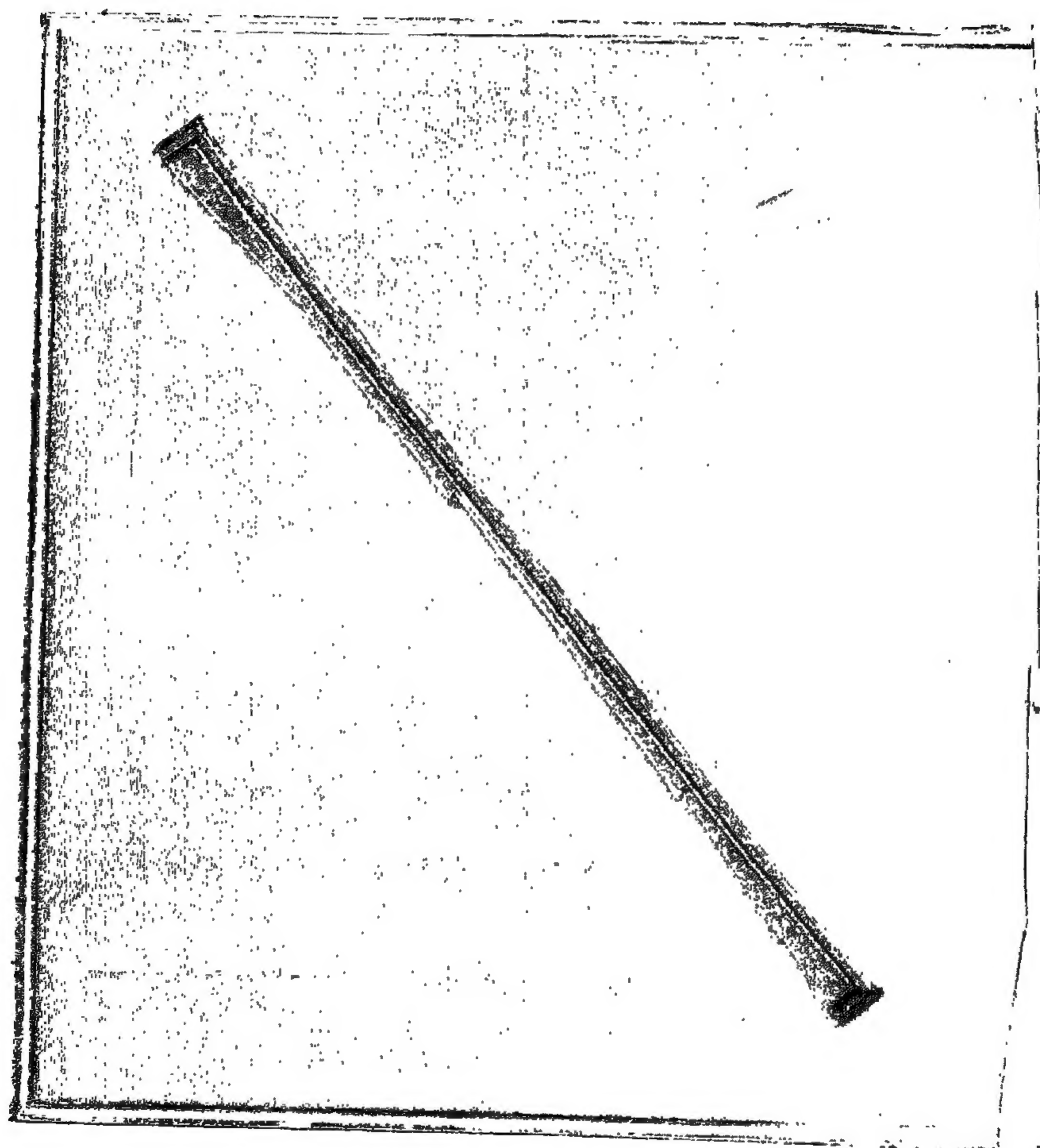
- ◆ محتويات كافة المتغيرات الفورية لهدف الخطأ Error Object.
- ◆ مجموعات ضبط الذاكرة (إجمالي المساحة الفارغة ، نطاق التشغيل ، توفر ذاكرة وخيالية وتوفر EMS).
- ◆ الوقت والتاريخ.
- ◆ الدليل الحالي ومساحة الأسطوانة المتوفرة .
- ◆ محتويات متغيرات بيئة PATH و COMSPEC و CLIPPER . ويمكنك تعديل البرنامج لعرض قيم متغيرات بيئة أخرى .

♦ مجموعة الاستدعاء العاملة عند حدوث الخطأ .

♦ وصف الشاشة عند حدوث الخطأ .

الخلاصة

باستخدام تركيب "بداية التسلسل..نهاية التسلسل" BEGIN
SEQUENCE..END SEQUENCE، وعبارة "إصلاح" RECOVER ووظيفة
ErrorBlock() ، ووظيفة "التنبيه" Alert() يمكننا إنشاء نظام إصلاح أخطاء حتى
ولو كانت خارجة عن سيطرتنا وإرادتنا.



- ❖ الكتاب الوحيد في العالم العربي الذي يشرح كليبر 5.2 ، بالإضافة إلى الكم الهائل من المعلومات التي يشرحها الكتاب من خلال أجزائه الثلاث.
- ❖ تم تقسيم الكتاب إلى ثلاث أجزاء هي: مقدمة البرمجة ، أساسيات البرمجة ، البرمجة المتقدمة.
- ❖ شرح لمعظم أوامر وتعليمات ووظائف كليبر الأساسية للإصدار 5.2.
- ❖ جزء خاص عن مقدمة البرمجة بلغة كليبر ماهي؟ وكيف يمكنك الاستفادة منها؟
- ❖ تصميم وإنشاء وكتابة أقوى التطبيقات الاحترافية باستخدام لغة كليبر 5.2.
- ❖ فصل موسع لطريقة استخدام برنامج كشف الأخطاء Debugger بأسلوب سهل وبسيط.
- ❖ كما خصص مقدار كبير من الجزء الثالث للحديث عن البرمجة باستخدام Tbrowser و TBColumn ومزاياها المفيدة في استعراض قواعد البيانات.
- ❖ كما تم شرح كتلة الشيفرة بأسلوب سهل ، يجعل من هذه التقنية الجديدة في كليبر مريحة وسهلة الاستخدام.
- ❖ كما خصص فصل للحديث عن استراتيجيات عمل الشبكة نوليل مع كليبر 5.2.
- ❖ فصل كامل للحديث عن مفاتيح الجمع والربط.
- ❖ شرح للمعالج الأولي Preprocessor وملفات الرويسة والموجهات وغير ذلك.
- ❖ فصل كامل لشرح طريقة إعلان المتغيرات بجميع أنواعها بأسلوب سهل وميسر.
- ❖ فصل كامل لشرح طريقة استخدام المصفوفات وكذلك الوظائف المتعلقة بها.
- ❖ شرح طريقة التحويل إلى نظام التشغيل MS-DOS باستخدام الرابط BLINKER 2.0
- ❖ شرح لطريقة تحويل برامجك من شيفرة المصدر source code إلى برامج قابلة للتنفيذ EXE. تعمل بشكل مستقل.
- ❖ فصل كامل يوضح طريقة تصميم واجهة المستخدم والأدوات التي يوفرها كليبر للقيام بهذه المهمة في توفير شكل جمالي ومريح للمستخدم.

كتاب



فترص

